Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica.

V-REP COMO HERRAMIENTA DE SIMULACIÓN DE SISTEMAS DINÁMICOS

Autor: Álvaro Morales Márquez
Tutor: José Ángel Acosta Rodríguez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

V-REP COMO HERRAMIENTA DE SIMULACIÓN DE SISTEMAS DINÁMICOS

Autor:
Álvaro Morales Márquez

Tutor:
José Ángel Acosta Rodríguez
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: V-REP COMO HERRAMIENTA DE SIMULACIÓN DE SISTEMAS DINÁMICOS

Autor: Álvaro Morales Márquez

Tutor: José Ángel Acosta Rodríguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos

Agradecimientos

Con este proyecto puedo decir que concluyo una etapa de mi vida. En los primeros cursos de carrera me fue demasiado bien. Sin embargo, acabé el que sería mi cuarto y último año abandonando el curso y una beca de colaboración debido a una fuerte depresión que me incapacitó para el desarrollo normal del curso.

Después de unos nueves meses tomando antidepresivas fuí capaz de retomar los estudios, me cambié de modalidad a electrónica y he superado todas las asignaturas sin demasiados problemas. Actualmente, un año depués de retomar los estudios me encuentro trabajando y fuera de casa de mis padres.

Todo este progreso como la realización de este proyecto no hubiera sido posible sin el apoyo incondicional de muchas personas. En primer lugar mi tutor, José Ángel, que desde que caí en la depresión me apoyo y me dió el tiempo que fuera necesario por si me veía capaz de retomar la beca de colaboración, cosa que no fue posible. También ha sido crucial el papel de mi familia, en especial de mi madre que ha estado en cada momento y para cada cosa que he necesitado, tanto en los buenos como en los malos momentos. Sin embargo, la persona que más me ha permitido evolucionar en todos los aspectos ha sido mi psicóloga Pilar, excelente tanto a nivel profesional como personal. Por último también debo estar agradecido de los amigos que tengo, tanto los que están día a día a mi lado como los que están lejos pero igualmente forman parte de mi vida.

Resumen

En este proyecto se ha analizado exhaustivamente el entorno de V-REP con la finalidad de desarrollar un simulador de robots dinámicos controlados mediante Matlab/Simulink. En primer lugar se analizan algunas funciones de la API y se establece una comunicación sincrona entre ambos programas. Una vez establecida, se pasa a analizar los elementos dinámicos más simples como muelles y motores mediante experimentación y validación con modelos matemáticos fiables. Con estos elementos ya bien definidos se realizarán algunos experimentos de sistemas poco complejos principalmente para demostrar como funcionan ambas herramientas y analizar algunas funcionalidades nuevas.

Después se pasa al análisis de sistemas más complejos, el brazo robótico *Alexandos* y un cuadrotor. Ambos robots son descritos paso por paso para poder realizar una copia en V-REP, para el cuadrotor incluso hay que realizar una hélice basada en el modelo matemático. El brazo, mediante un control adaptativo, llevará a cabo unos experimentos para ejercer una fuerza determinada sobre un muelle y el UAV, mediante un control *Backstepping* realizará unas pruebas de vuelo.

Índice

1 OBJETIVOS Y ALCANCE DEL PROYECTO

En este capítulo se describen los objetivos y el alcance del proyecto para tener una visión global del trabajo realizado.

Los objetivos principales de este trabajo son:

-Realizar una comuniación fiable entre Matlab[1] y V-REP[2].

-Implementación detallada de la API de V-REP para Matlab[3].

-Analizar los elementos básicos de V-REP (muelle, motor).

-Simular y controlar sistemas dinámicos simples.

-Modelar el brazo *Alexandros*[4] en V-REP y controlarlo mediante Matlab.

-Ejercer un control de fuerza con el brazo *Alexandros*.

-Diseño y validación de una hélice.

-Modelar un UAV en V-REP y controlarlo[5] mediante Matlab.

-Realizar pruebas simples de vuelo con el UAV.

El análisis exahustivo de los modelos dinámicos de mayor complejidad así como un análisis teórico de los controladores que van a ser usados a lo largo del proyecto quedan fuera del alcance del proyecto.

2 SINCRONIZACIÓN ENTRE MATLAB Y V-REP

En este capítulo se explica en detalle como sincronizar Simulink y V-REP, incluyendo códigos, diagramas de flujo y explicaciones paso por paso.

2.1 Necesidad de sincronización

Primero es necesario establecer una comunicación entre Matlab [1], que calcula y lleva a cabo la tarea de control, y V-REP [2], que simula la dinámica del sistema. Para ello V-REP tiene scripts y funciones que facilitan bastante el trabajo sin embargo hay que asegurarse de que la comunicación se realiza de forma estable y síncrona.

Un problema que se plantea al conectar dos programas por separado (V-REP y Matlab) es que cada simulación se ejecuta a una frecuencia diferente, en los primeros experimentos se llevaron a cabo mediante una comunicación asíncrona y los resultados no fueron satisfactorios. Quizás en alguna aplicación de más alto nivel no importe pero en este proyecto al tratarse de algoritmos de control, la sincronización es algo crítico y por tanto todas la comunicaciones se llevaran a cabo de forma síncrona como se explicará a continuación.

2.2 Descripción del método de sincronización

La solución para que funcione de manera síncrona con Simulink es el uso de un script principal que ejecuta tanto Simulink como V-REP y uno secundario que va ejecutándose en cada paso de simulación. Ambos scripts tienen una comunicación entre ellos mediante variables globales.

El funcionamiento de manera resumida es el siguiente:

- En primer lugar establece la comunicación y empiezan tanto las simulaciones en Simulink como V-REP.
- Una vez inicializados, el proceso ejecuta un paso de simulación en Simulink, recoge las señales de actuación y para Simulink.
- Envia estas señales a V-REP y simula un paso de simulación, recoge las medidas y las envía a Simulink, parándose.
- Una vez llegue a Simulink este vuelve a ejecutarse, creando un ciclo hasta que acabe la simulación.

Aunque es más complejo que el método asíncrono nos asegura una comunicación fiable.

A continuación se describe en detalle el método para que funcionen Matlab y V-REP de manera síncrona:

1. Declarar las variables de manera global para que cualquier función pueda acceder a ellas. Estas variables son el ID del cliente y aquellas articulaciones o elementos que actúan en ambos programas.

```
global clientID;   %variable global con el cliente
global j1;   %variables globales que se asociarán a los elementos de V-REP
global j2;
```

2. Asegurarse de que no hay ninguna conexión establecida y entonces iniciar la comunicación mediante el puerto 19997 (asignado por defecto).

```
disp('Program started');
```

```
vrep=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
vrep.simxFinish(-1); % just in case, close all opened connections
clientID=vrep.simxStart('127.0.0.1',19997,true,true,5000,5);
if (clientID>-1)
  disp('Connected to remote API server');
```

3. Establecer la simulación como síncrona y empezar la simulación.

```
% enable the synchronous mode on the client:
 vrep.simxSynchronous(clientID,true);
 % start the simulation:
 vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait);
```

4. Declarar los objetos, mediante estas funciones se comunica la variable en Matlab con el objeto en V-REP. En este caso se comunican las variables j1 y j2 con los objetos joint1 y joint2.

```
[r1,j1]=vrep.simxGetObjectHandle(clientID,'joint1#',vrep.simx_opmode_blocking)
;

[r2,j2]=vrep.simxGetObjectHandle(clientID,'joint2#',vrep.simx_opmode_blocking)
;
```

5. Ejecutar la simulación en Simulink.

```
 sim('sistema');
```

6. Cierra el simulador en V-REP.

```
        % stop the simulation:
        vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait);


        % Now close the connection to V-REP:
        vrep.simxFinish(clientID);
```

7. En el caso de que la simulación no hubiese funcionado aparecería esto en Matlab.

```
    else
        disp('Failed connecting to remote API server');
    end
    vrep.delete(); % call the destructor!
    disp('Program ended')
```

En el caso de no necesitar Simulink (algo que no es lo buscado, pero puede interesar en algún caso) se puede usar un solo script que dirija tanto la inicialización como los pasos de simulación mediante un bucle.

Una vez dentro de Simulink, debemos crear una función que haga de interfaz entre Matlab y V-REP. Los pasos

a seguir son los siguientes:

a. Las entradas (in) de la función son los datos que debe recibir V-REP.

```
function [ out ] = step( in )
          vel1=in(1); %en este caso una velocidad
```

b. Declarar en esta función las variables globales que fueron creadas anteriormente.

```
global clientID;
global cuerpo; %variables globales que se asociarán a los elementos de V-REP
global motor;
```

c. Llamar a la Api:

```
vrep=remApi('remoteApi');
```

d. Enviar los datos a V-REP (por lo general, la señal de control):

```
                vrep.simxSetJointTargetVelocity(clientID,     motor,     vel1,
vrep.simx_opmode_oneshot_wait);
```

e. Avanzar un paso de simulación en V-REP:

```
 vrep.simxSynchronousTrigger(clientID);
```

f. Leer los datos de V-REP:

```
            [s,posicion]=vrep.simxGetObjectPosition(clientID,      cuerpo,-1,
vrep.simx_opmode_streaming); %Vector con las componentes espaciales[x,y,z]
            [s,orientacion]=vrep.simxGetObjectOrientation(clientID,  cuerpo,-
1, vrep.simx_opmode_streaming); %Vector con las componentes angulares [α,β,γ]
```

g. Actualizar las salidas (out) de la función:

```
posx=posicion(1); %Separar el vector es sus tres componentes
posy=posicion(2);
posz=posicion(3);
alpha=orientacion(1); %Separar el vector es sus tres componentes
beta=orientacion(2);
gamma=orientacion(3);

out=[double(posx),double(posy),double(posz),double(alpha),double(beta),doubl
e(gamma)]; %Devuelve un vector con las 6 componentes (posición y orientación)

end
```

Las funciones usadas en ambos scripts vienen detalladas en la documentación de la API [3]. En la Figura 1 se puede observar un diagrama que resume la sincronización entre ambos programas.

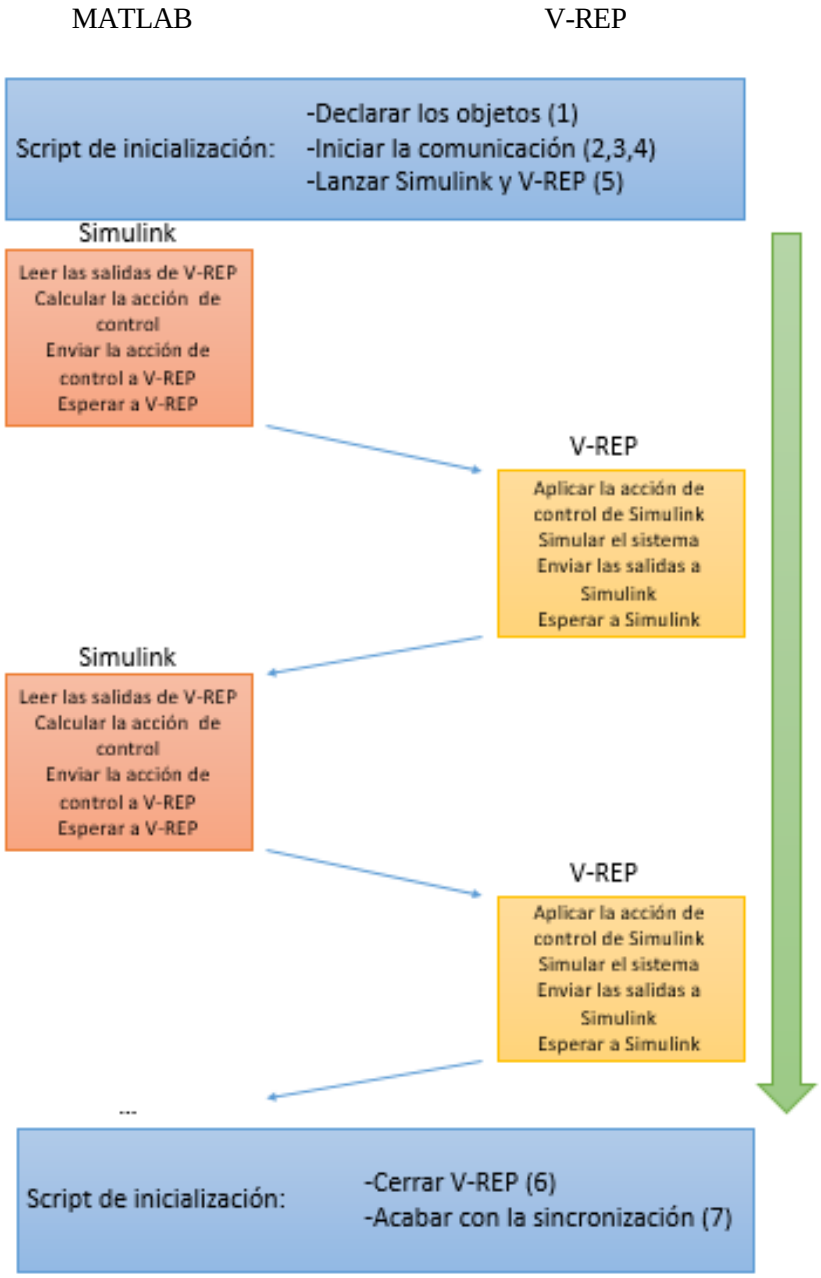

Figura 1: Diagrama de la sincronización entre Matlab/Simulink y V-REP

3 IDENTIFICACIÓN DE ELEMENTOS BÁSICOS Y SUS PROPIEDADES

3.1 Introducción a V-REP

3.1.1 ¿Qué es V-REP?

V-REP es un software desarrollado por Coppelia Robotics que permite la simulación de robots. Tiene una versión educativa, que es con la que se trabajará en este proyecto, y otras de pago que permiten una mayor funcionalidad. Cuenta con bastante información tanto en el manual de usuario como en foros, además de tutoriales para familiarizarse con el entorno del simulador, sin embargo en muchas ocasiones no fue suficiente.

3.1.2 Enfoque de V-REP en el proyecto

El uso principal que se le va a dar al software es simular el comportamiento dinámico de los diferentes sistemas con los que se trabajará. Es un programa bastante versátil que permite comunicarse con nodos de ROS, plugins o APIs remotos.

V-REP trabaja en un entorno gráfico que se basa en jerarquías como se puede observar en la Figura 2. Cuenta con una amplia gama de robots ya creados y funcionalidades de alto nivel, pero no van a ser utilizados y se van a crear los modelos desde el princpipio. En es proyecto se irá describiendo como se crean los modelos y su verificación comparando su respuesta frente a alguna simulación en Matlab. Los elementos que se van a analizar en este capítulo son motores, muelles y hélices.

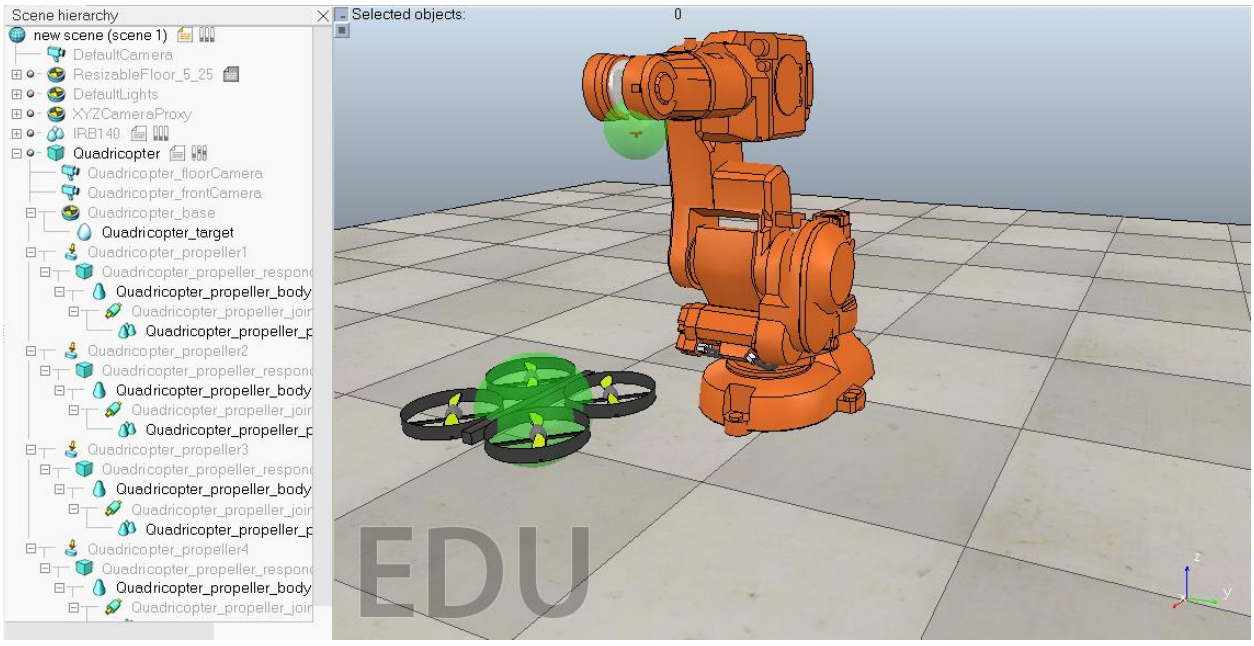

Figura 2: Interfaz de V-REP

3.2 Elementos básicos de V-REP

3.2.1 Motor

Los motores aparecen en la mayoría los sistemas de robótica y es importante tenerlos bien caracterizados. Principalmente se necesita conocer la inercia del rotor (Jm) y el efecto viscoso (B). La ecuación que modela al

motor es la siguiente:

Torque = (Jm+ J_{link})*aceleración+B*velocidad.

Siendo J_{link} la inercia del eslabón. Para los experimentos vamos a usar un modelo muy simple, formado por un motor, un eslabón fijo y uno móvil, como el de la Figura 3.

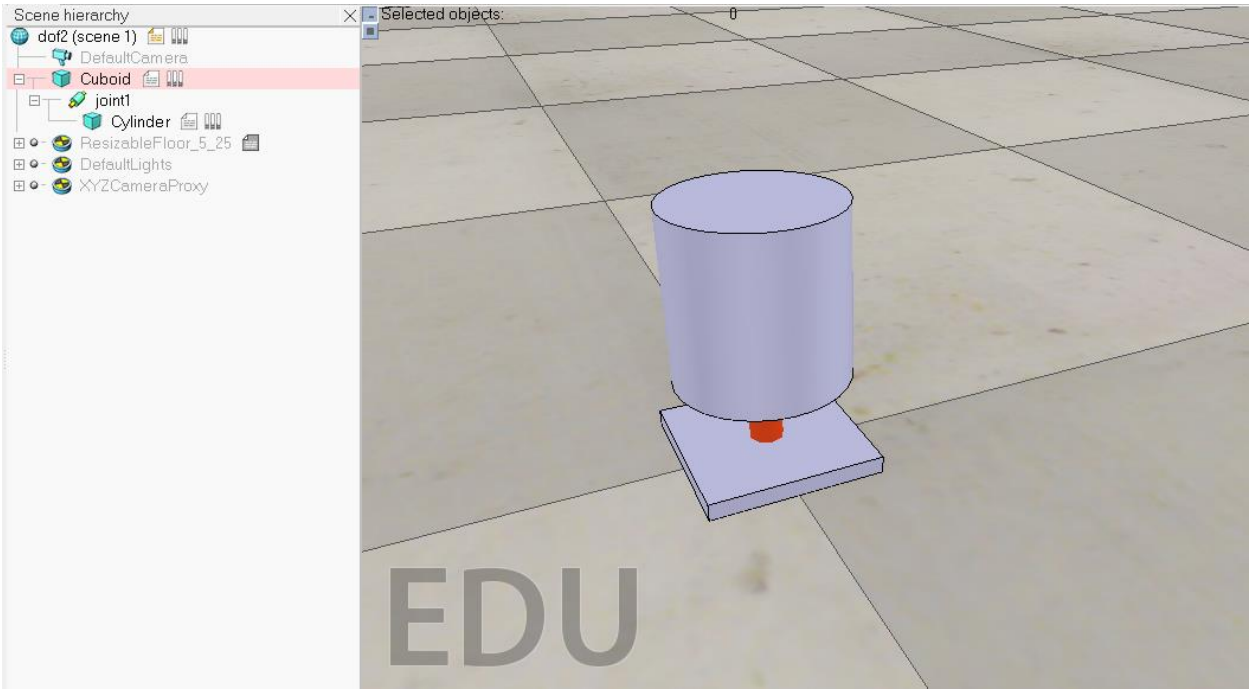

Figura 3: Sistema para el análisis del motor.

Para calcular el modelo dinámico se van a realizar dos experimentos, con una inercia del eslabón de 1e-12 de tal manera que la suma de ambas inercias debe ser del orden de la inercia del rótor.

Primero se introduce un escalón en velocidad:

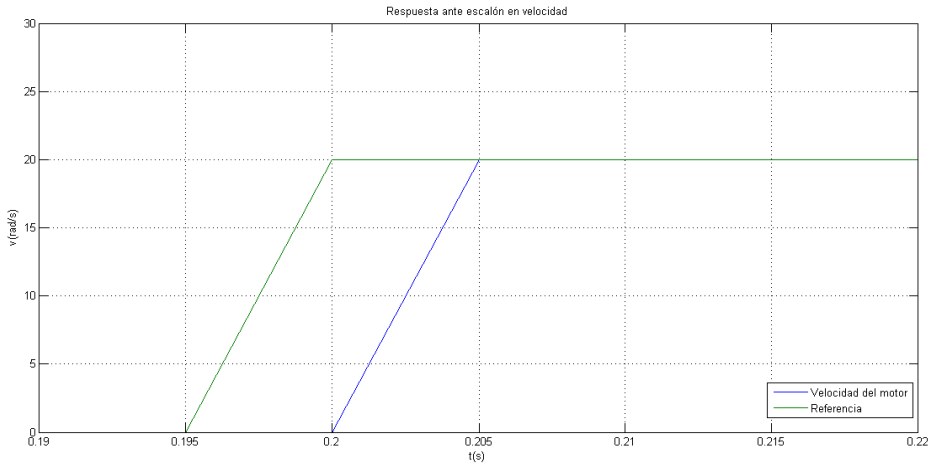

Figura 4: Experimento 1 de la dinámica del motor

En la Figura 4 se puede ver que tarda menos de 5ms en subir, es decir el paso de integración mínimo con el que trabaja V-REP (la pendiente es la interpolación de Matlab, debería ser un escalón), siendo una gran aproximación.

El segundo experimento es una rampa (unitaria) en velocidad:

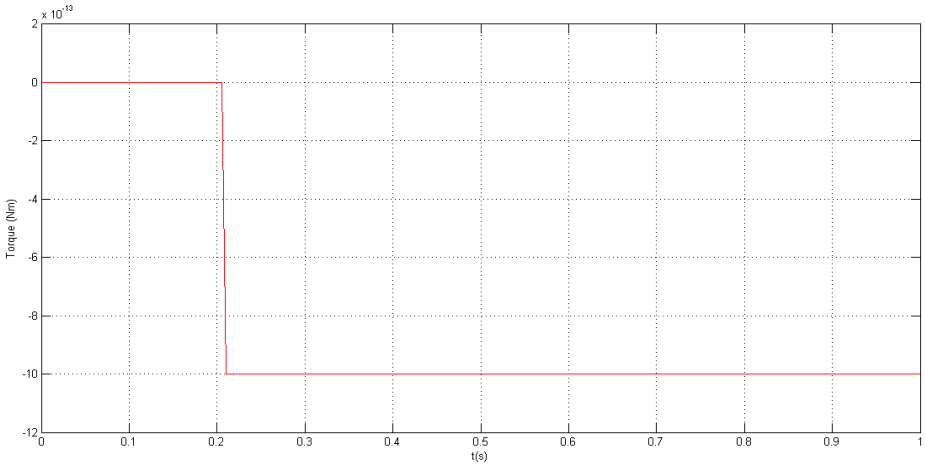

Figura 5: Experimento 2 de la dinámica del motor

En la Figura 5 se observa que el sistema tendría una aceleración de valor 1. Como se puede observar el torque es 1e-12, es decir el producto de la aceleración por la inercia del eslabón (aun siendo muy pequeño) solapan a los términos Jm y B. Por tanto podemos concluir que el motor básicamente va a ser ideal en la práctica y el par que crea solo debe vencer a la inercia del eslabón. La ecuación dinámica quedaría:

Torque =J_{link}*aceleración.

Si por algún motivo interesase introducir una pequeña dinámica, esta debería ser introducida desde simulink. En la Figura 6 se muestra el diagrama de bloques del primer experimento y Figura 7 el diagrama del segundo experimento.

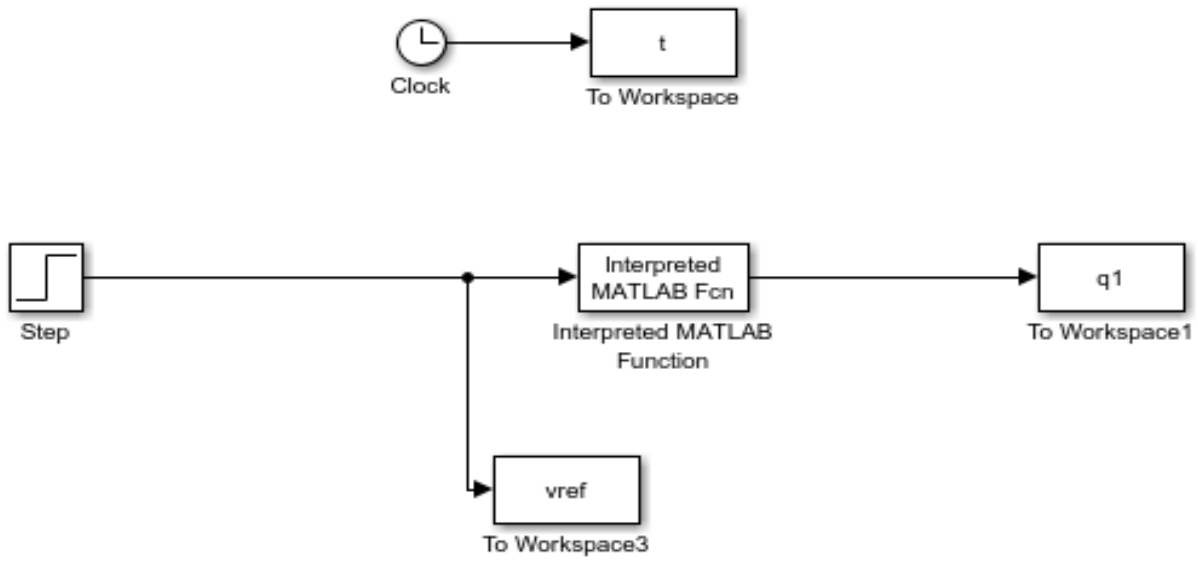

Figura 6: Esquema de simulink para el primer experimento de la identificación del motor

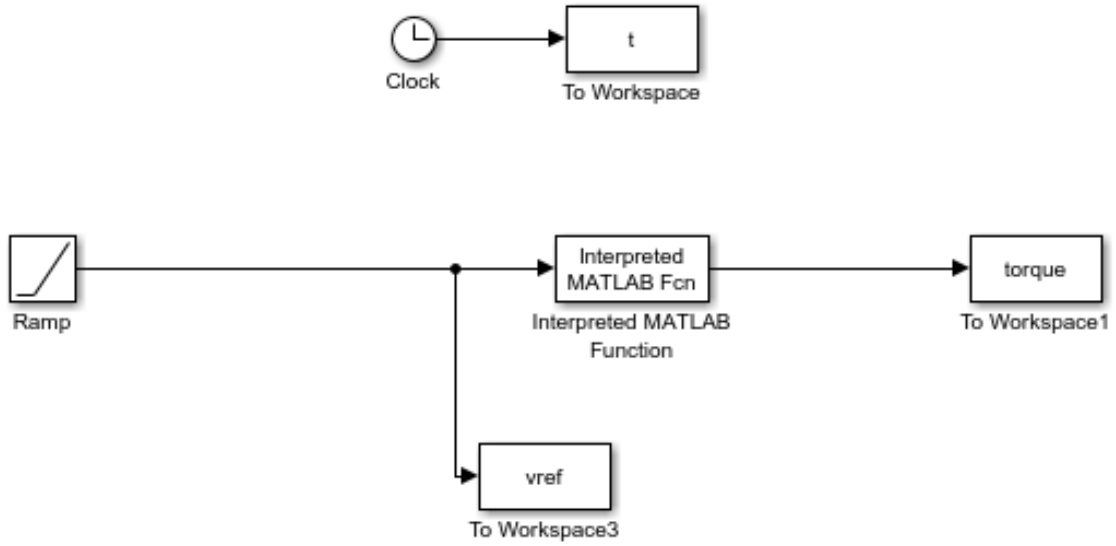

Figura 7: Esquema de simulink para el segundo experimento de la identificación del motor

El bloque "Interpreted MATLAB Fcn" ejecuta el siguiente script

```
function [ out ] = step( in )
            vel1=in(1); %velocidad del eje

global clientID;    %variables globales que le llegan desde SINCRO.m
global j1;

vrep=remApi('remoteApi');
           %Envia la velocidad
            vrep.simxSetJointTargetVelocity(clientID,        j1,        vel1,
vrep.simx_opmode_oneshot_wait);
           %Lee la posición
             [s1,        pos1]=vrep.simxGetJointPosition(clientID,        j1,
vrep.simx_opmode_streaming);
           %Lee la fuerza aplicada

[b,v]=vrep.simxJointGetForce(clientID,j1,vrep.simx_opmode_streaming);
           %Ejecuta V-REP
                vrep.simxSynchronousTrigger(clientID);

           %Devuelve posición y fuerza
              out=[double(pos1),double(v)];

end
```

Para introducir un motor en V-REP: Add → Joint → Revolute/Prismatic (dependiendo del tipo que se necesite). En la Figura 8 se muestras las propiedades dinámicas del motor, hay que marcar la casilla de motor hablitado y dejar el motor en modo Torque/force mode. También puede modificarse el par máximo si interesa.

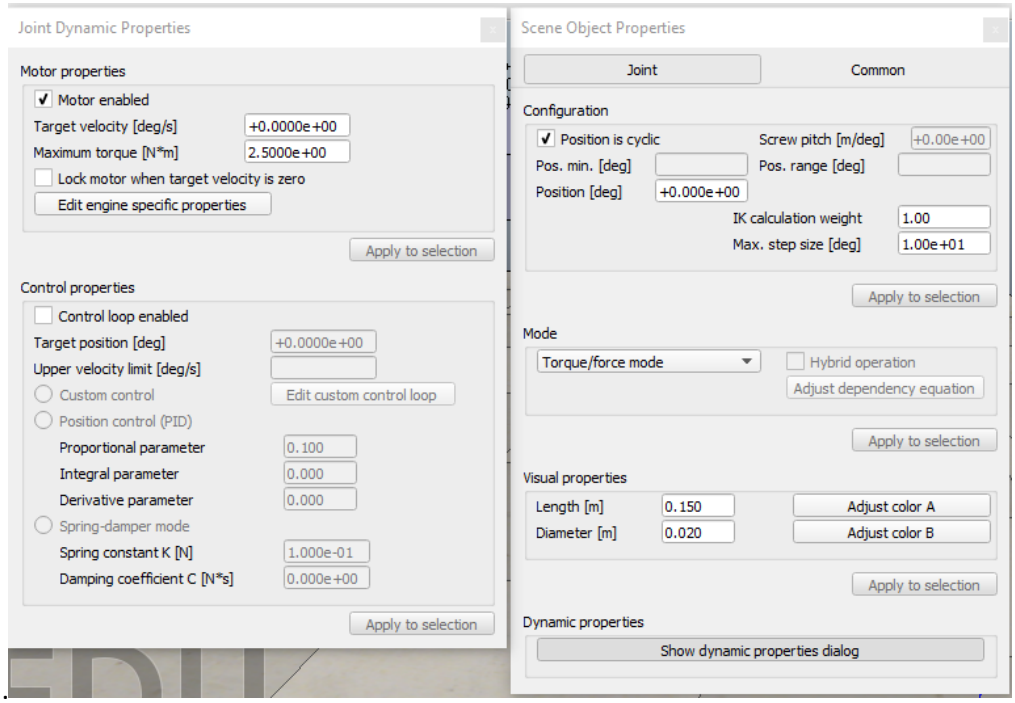

Figura 8: Propiedades dinámicas del motor en V-REP

Para crear un elemento pasivo (como los eslabones) en V-REP: Add→Primitive Shape→ Cuboid/Cylinder (o la forma que interese). Una vez creado el elemento podremos desplegar los paneles de la Figura 9. En la pestaña View/Modify geometry se puede modificar la forma para ajustarlo al tamaño que interese. Las propiedades dinámicas (panel izquierdo) permiten modificar la masa y la inercia, además podemos hacer que el cuerpo sea fijo desmarcando Body is responsable y Body is dinamic para usar el eslabón como punto fijo de referencia.

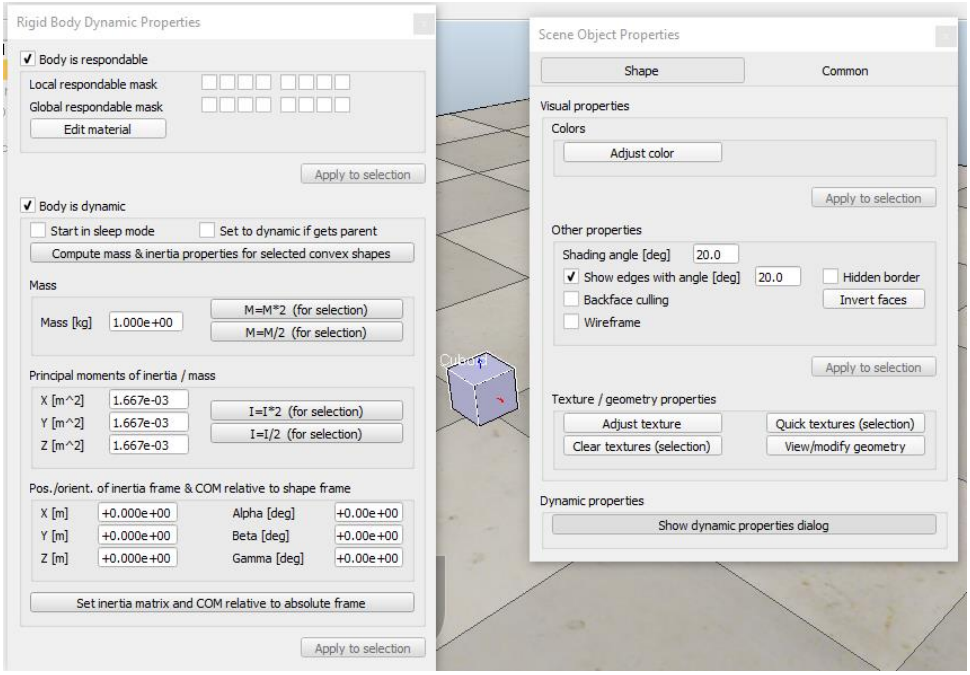

Figura 9: Propiedades de un elemento pasivo

3.2.2 Muelle

V-REP no dispone de un elemento "muelle" como tal. En su lugar se va a usar un elemento motor (revolute joint) con un controlador que imita el comportamiento de un muelle. Como no es un muelle ideal vamos a

comparar las ecuaciones analíticas de un caso simple con la simulación del mismo modelo. El modelo es como se muestra en la Figura 10, formada por una esfera fija (apoyo), un muelle y un cilindro que hace de masa.

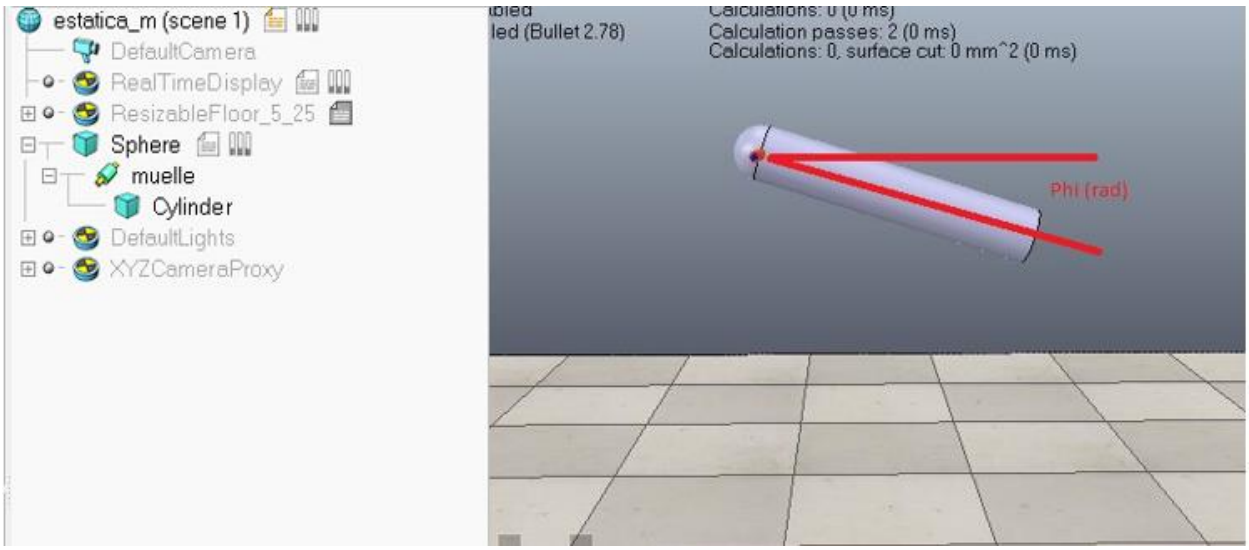

Figura 10: Experimento de las características del muelle.

Como se puede observar no es más que un eslabón anclado a un muelle que se deja caer y forma un ángulo (φ) con la horizontal. La ecuación que modela el sistema es:

$$I\ddot{\varphi} + C\dot{\varphi} + K\varphi + \frac{1}{2}mgcos(\varphi) = 0$$

Donde I es la inercia, m es la masa, g la constante gravitacional y K y C las constantes del muelle.

3.2.2.1 Caraterística estática

Para comprobar que el muelle tiene un comportamiento correcto en régimen permanente usando el modelo anterior se va a comparar el ángulo formado una vez terminado el transitorio variando la constante K del muelle. En V-REP simplemente se deja que la simulación avance y se puede ver fácilmente el ángulo y este ángulo obtenido se compara con los valores de un modelo teórico. En la Figura 11 se puede observar que apenas difieren el modelo teórico y los experimentos:

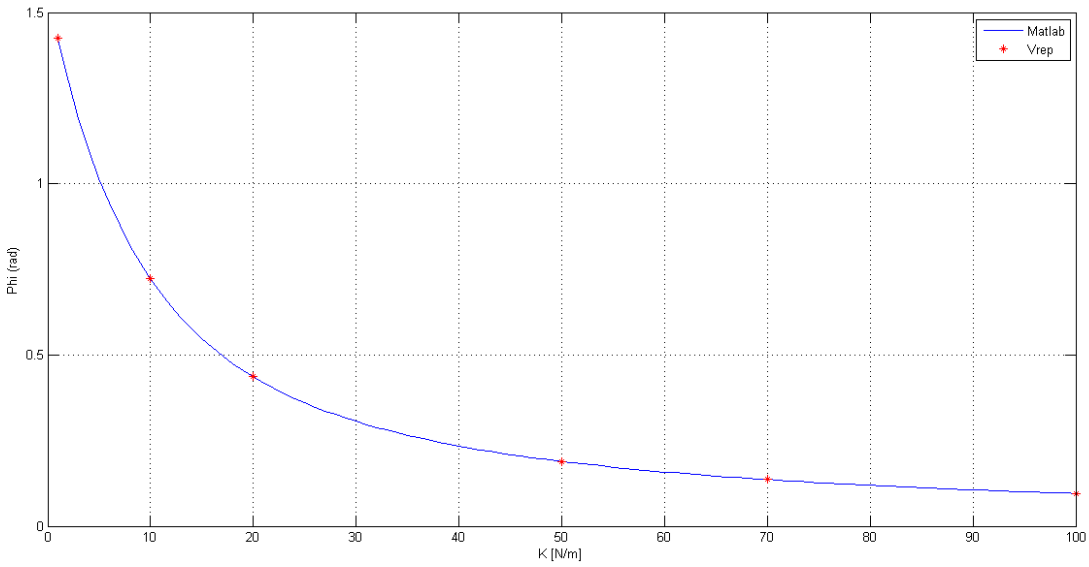

Figura 11: Comparativa de la característica estática

Para calcular esta dinámica estática en V-REP, basta con dejar que el sistema evolucione y ver cuanto se ha desplazado en eslabón, no es necesario ni sincronizarlo con Simulink. Para obtener los resultados en Matlab se usaron los siguientes códigos:

```
%Código para calcular el ángulo formado por el eslabón en régimen
%permanente. Con una k del muelle variando entre 1 y 100
for k=1:100
    f=@(x)F(x,k);
   a(k)= fsolve(f,0);


end
k=1:100;
plot(k,a);grid;
```

```
function F=basicfun(x,k)
%Ecuación del muelle en régimente permanente
F=cos(x)-x*0.1039*k;

end
```

3.2.2.2 Característica dinámica

Este experimento es similar al anterior, aunque aquí se verificará que el modelo de V-REP cumple la evolución temporal teórica. El modelo teórico fue obtenido con una simple simulación en Simulink.

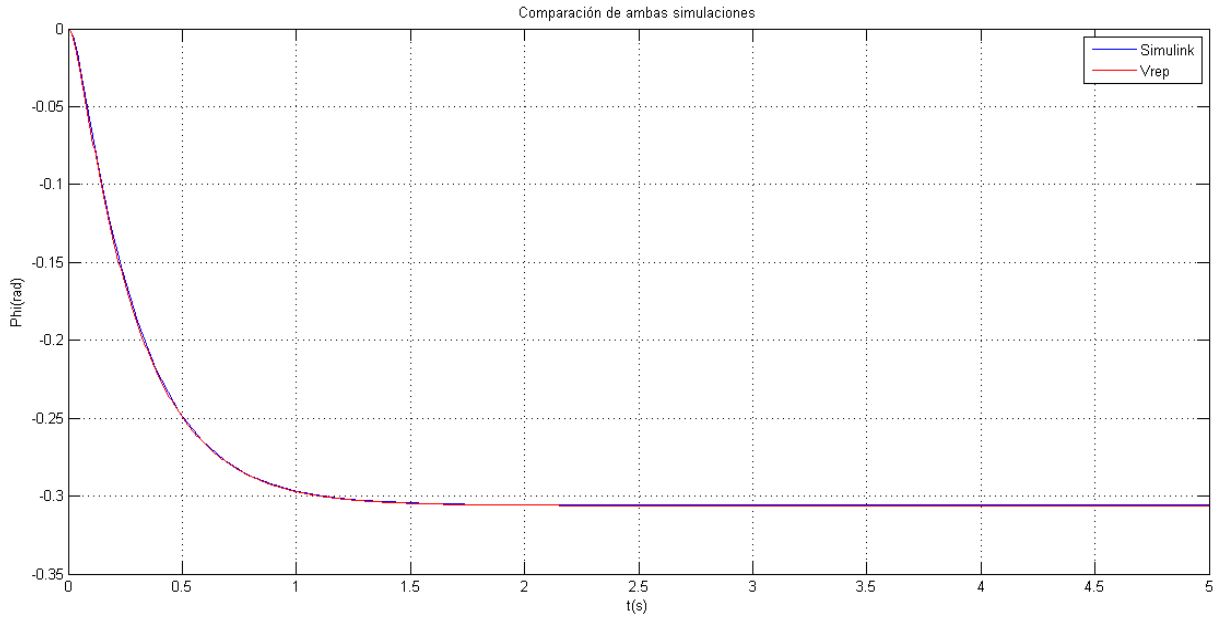

Figura 12: Comparativa de la característica dinámica.

Como se puede observar en la Figura 12, la diferencia entre ambas gráficas es casi inapreciable, por tanto podemos deducir que el muelle tiene una dinámica como la esperada en el modelo dinámico.

Para introducir un muelle se parte del motor explicado anteriormente pero se marcan las casillas de Control Loop Enable y Spring-Damper mode. Además se habilitarán las casillas para rellenar con las constantes K y C del muelle.

Para reproducir el experimento, en la Figura 13 se muestra el esquema de simulink seguido del código que calcula la dinámica del sistema.

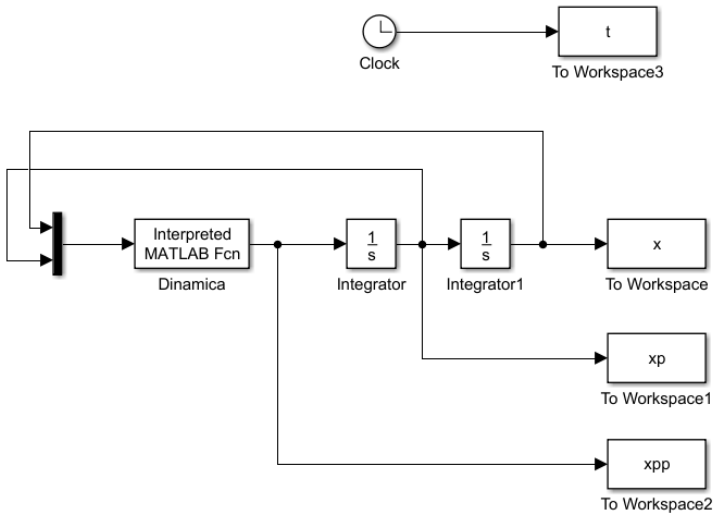

Figura 13: Diagrama de bloques de simulink para la dinámica de un muelle

```
function [ out ] =dinamica( in )
%Unidades en SI
x=in(1); %posición
xp=in(2); %velocidad
I=0.3297; %inercia
m=3.927; %masa
l=0.5; %longitud del eslabón
g=9.8; %gravedad
k=30; %constante del muelle
c=10; %constante de amortiguamiento

xpp=(-k*x-c*xp-m*g*cos(x)*l/2)/I; %Aceleración del eslabón
out=xpp;


end
```

4 SIMULACIONES SIMPLES COMPARANDO SIMULINK Y V-REP

En este capítulo se va a realizar una comparativa entre Simulink y V-REP, analizando simulaciones con controladores básicos. Se analizarán dos robots articulados. El primero compuesto por dos articulaciones y uno segundo que además de esas dos articulaciones incluye un muelle.

4.1 Robot 2dof con control en posición

En este apartado se va a simular el sistema de la Figura 14 en posición tanto en V-Rep como en Matlab. Está formado por por dos eslabones móviles, uno fijo y dos motores, los cuales se describen en el capítulo anterior.

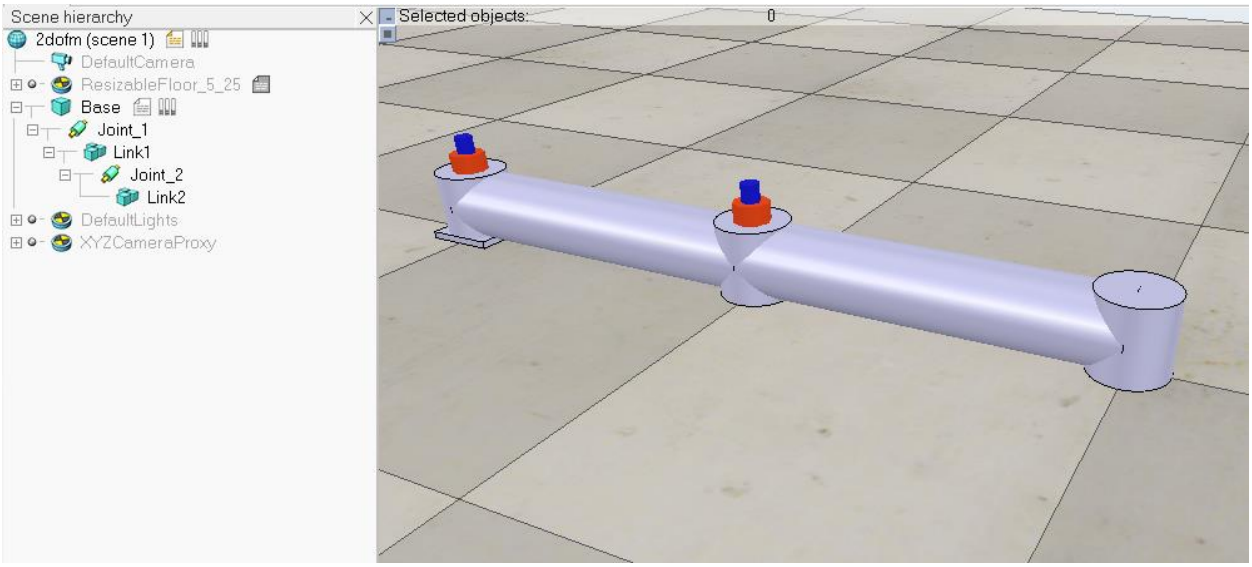

Figura 14: Jerarquía y modelo 3D de un robot 2dof.

El controlador en posición de la Figura 15 es simplemente proporcional al error, el bloque "subsystem" representa a Maltab o V-rep, los demás bloques no varían al realizar las simulaciones. En el caso de V-rep simplemente es la función de sincronización adapatada a este caso particular. Se usan bloques de "zero-order hold" y "unit delay" para asegurar el tiempo de muestreo y reducir los bucles alegebraicos en el lazo de control.

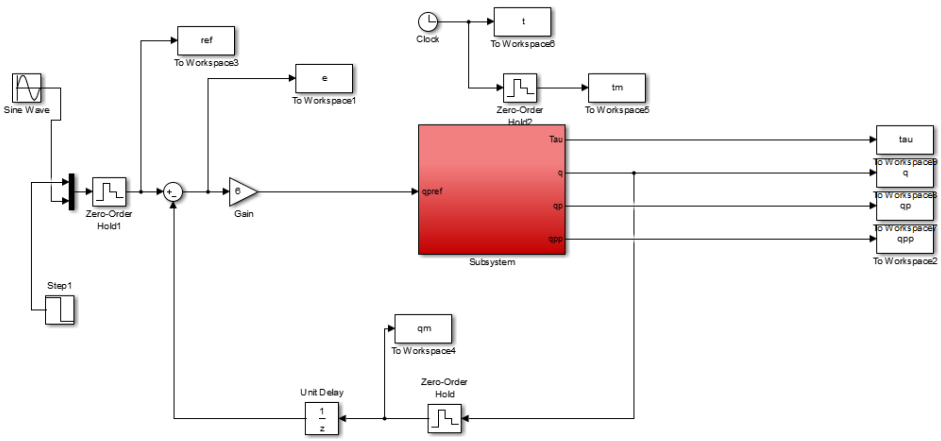

Figura 15: Modelo del control en posición en Simulink

A continuación se describe en detalle el diagrama en Simulink y el cógido usado.

El subsistema en Matlab es como se observa en la Figura 16, se puede ver un lazo en velocidad con un PID y un pequeño filtro de paso bajo.

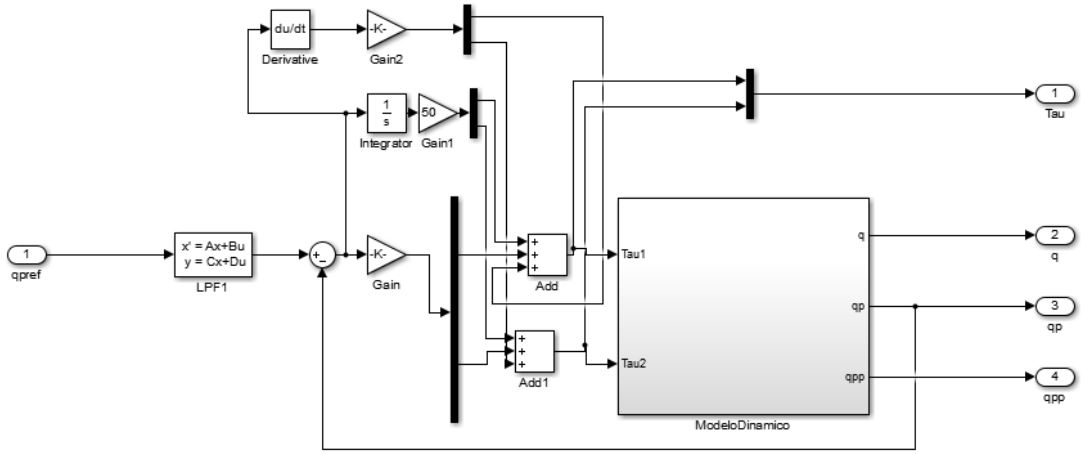

Figura 16: Lazo de velocidad para modelar el motor.

El modelo dinámico a su vez está formado por los elementos que se pueden observar en la Figura 17.

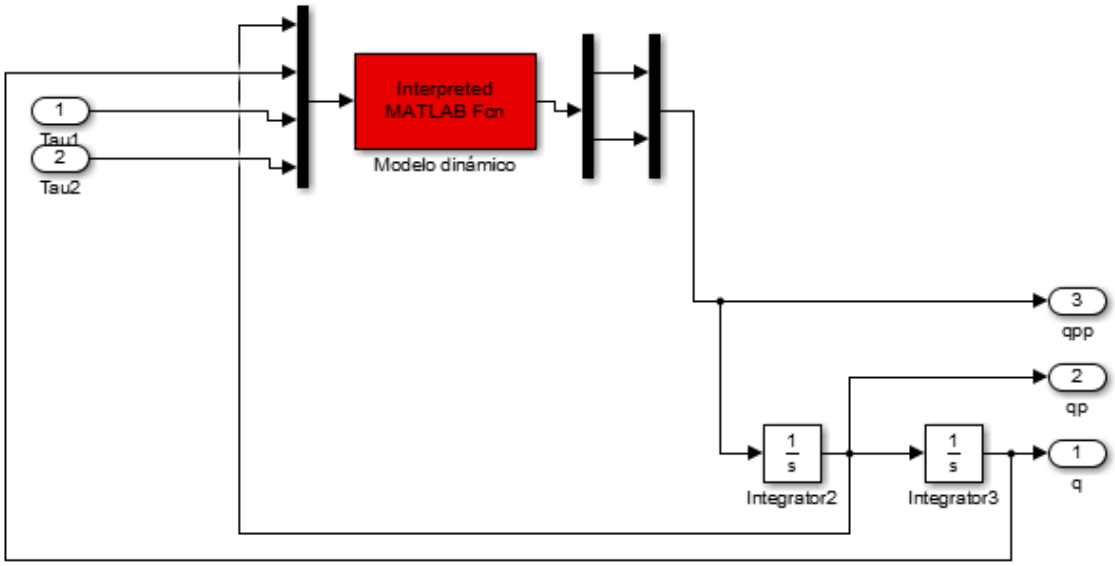

Figura 17: Modelo dinámico del motor

La ecuación dinámica se obtiene a partir de resolver la siguiente expresión:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})$$

Donde M es la matriz de inercias del sistema y C la matriz de los términos de Coriolis. La función en Matlab que implementa el cálculo de la dinámica es la siguiente:

```
function [ out ] = dinamica( in )
qp1=in(1);
qp2=in(2);
```

```
q1=in(3);
q2=in(4);
Tau(1,1)=in(5);
Tau(2,1)=in(6);
M =[  2.128*cos(q2) + 2.811, 1.064*cos(q2) + 0.5338;
    1.064*cos(q2) + 0.5338,                  0.5338];
C =[-1.064*sin(q2)*qp2^2-2.128*qp1*sin(q2)*qp2;
    1.064*qp1^2*sin(q2)];
 qpp=M\(Tau-C);
 out=[double(qpp(1)),double(qpp(2))];
     end
```

El resultado de comparar ambos modelos se puede ver en la Figura 18. Se observa como el resultado es satisfactorio y las direferencias entre ambos modelos son casi inapreciables. El controlador de posición no está muy elaborado ya que no es el objetivo de esta sección.

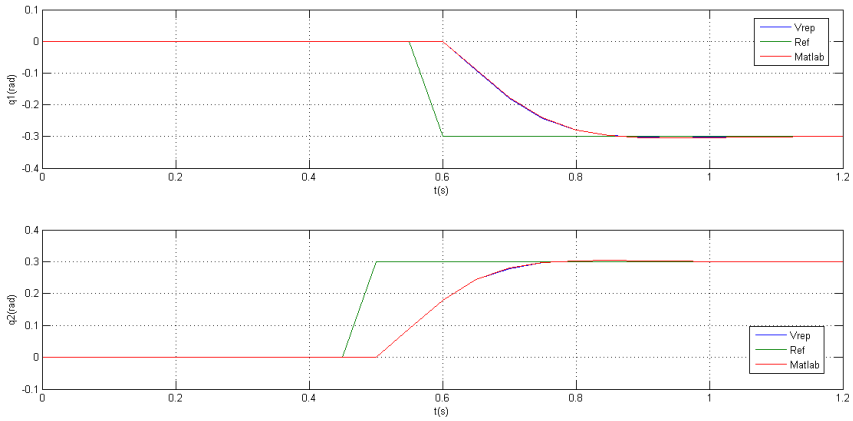

Figura 18: Comparativa del control en posición de V-REP y Simulink.

4.2 Robot 2dof con un muelle

En este apartado se va a analizar el comportamiento conjunto entre motores y muelles mediante un robot flexible. Para ellos tenemos el modelo de la Figura 19, similar al anterior de dos grados de libertad pero entre el eslabón dos y el motor dos tiene un pequeño eslabón con un muelle. De nuevo se comparará la respuesta de Matlab y V-REP.

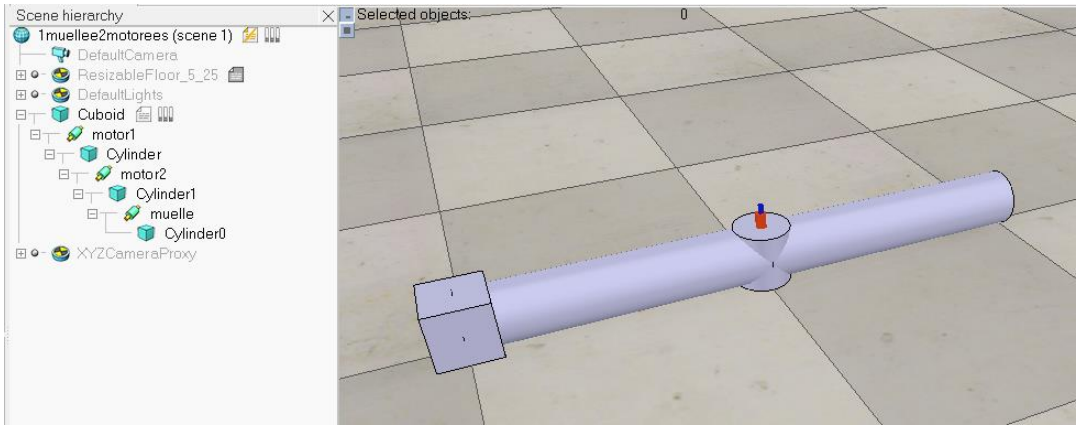

Figura 19: Jerarquía y modelo 3D de un robot 2dof con un muelle.

4.2.1 Control en velocidad

En este apartado se va a comparar el modelo de V-rep y Matlab ante un escalón en velocidad, el modelo en Matlab es el de la Figura 20 (análogo al del apartado sin muelles):

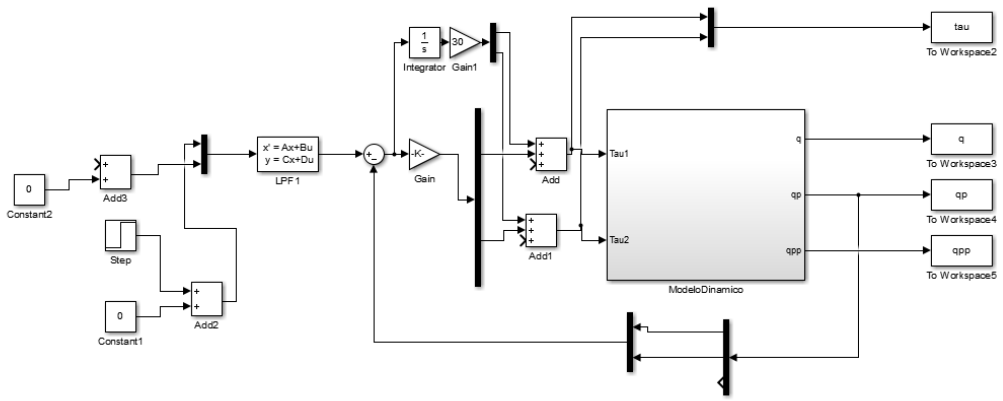

Figura 20: Esquema del control en velocidad.

La ecuación dinámica se obtiene a partir de resolver la siguiente expresión:

$$\tau = M(q)\ddot{q} + C(q,\dot{q}) + U(q)$$

Donde M es la matriz de inercias del sistema, C la matriz de los términos de Coriolis y U la matriz de las fuerzas conservativas (muelles). La función en Matlab que implementa el cálculo de la dinámica es la siguiente:

```
function [ out ] = dinamica( in )
qp1=in(1);
qp2=in(2);
qp3=in(3);
q1=in(4);
q2=in(5);
q3=in(6);
Tau(1,1)=in(7);
Tau(2,1)=in(8);
Tau(3,1)=0;
k=10;
d=0.2;
M =[   0.9818*cos(q2 + q3) + 1.772, 0.4909*cos(q2 + q3) + 0.336, 0.4909*cos(q2
+ q3) + 0.3297;
0.4909*cos(q2 + q3) + 0.336, 0.336,   0.3297;
0.4909*cos(q2 + q3) + 0.3297, 0.3297, 0.3297];
C =[    -0.4909*sin(q2 + q3)*(qp2 + qp3)*(2.0*qp1 + qp2 + qp3);
                                    0.4909*qp1^2*sin(q2 + q3);
                        0.4909*sin(q2 + q3)*qp1^2 + 1.0*d*qp3];
U=[0;0;k*q3];
```

```
qpp=M\(Tau-C-U);

out=qpp;

     end
```

Y en el caso de usar V-REP, la función de sincronización tiene la siguiente forma:

```
function [ out ] = step( in )
            vel1=in(1); %velocidad de los ejes 1 y 2 que llegan desde simulink
            vel2=in(2);

global clientID;    %variables globales que le llegan desde SINCRO.m
global j1;
global j2;
global j3;

 vrep=remApi('remoteApi');
%Establecer la velocidad en cada motor (acción de control)
            vrep.simxSetJointTargetVelocity(clientID,       j1,       vel1,
vrep.simx_opmode_oneshot_wait);
            vrep.simxSetJointTargetVelocity(clientID,       j2,       vel2,
vrep.simx_opmode_oneshot_wait);


%Obtener la posición de cada articulación
            [s1,        pos1]=vrep.simxGetJointPosition(clientID,       j1,
vrep.simx_opmode_streaming);
            [s2,        pos2]=vrep.simxGetJointPosition(clientID,       j2,
vrep.simx_opmode_streaming);
             [s2,        pos3]=vrep.simxGetJointPosition(clientID,       j3,
vrep.simx_opmode_streaming);
%Obtener la fuerza ejercida en cada articulación
             [s2,          tr1]=vrep.simxGetJointForce(clientID,         j1,
vrep.simx_opmode_streaming);
            [s2,          tr2]=vrep.simxGetJointForce(clientID,         j2,
vrep.simx_opmode_streaming);
            [s2,          tr3]=vrep.simxGetJointForce(clientID,         j3,
vrep.simx_opmode_streaming);
%Trigger para que actúe V-REP
               vrep.simxSynchronousTrigger(clientID);



%Variables de salida: posición angular de los tres motores y las fuerza
ejercida

out=[double(pos1),double(pos2),double(pos3),double(tr1),double(tr2),double(t
r3)];

end
```

Mediante esta función, se fijan las velocidades de los motores 1 y 2 y se leen la posición y la fuerza ejercida en las 3 articulaciones.

Ante un escalón en velocidad obtenemos los resultados que se muestran en la Figura 21, donde se observa que tienen un comportamiento idéntico excepto la dinámica del muelle que difiere un poco en el transitorio del principio hasta que alcanza un régimen permanente.

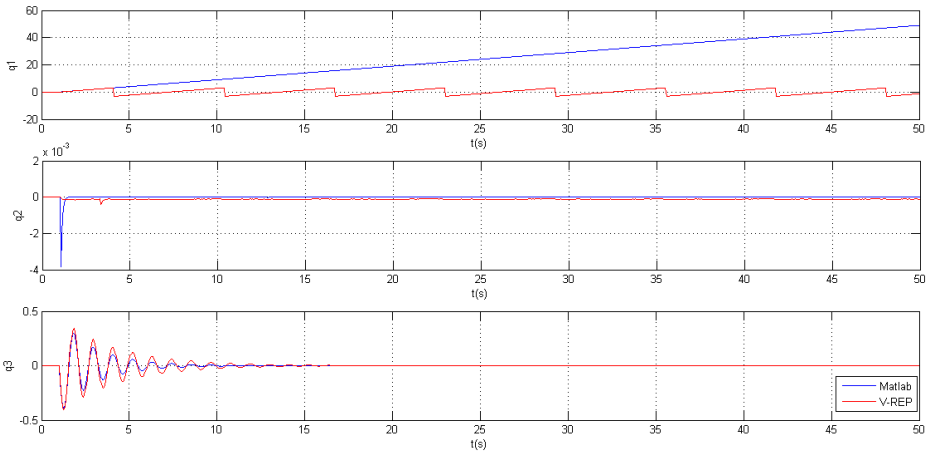

Figura 21: Escalón en velocidad.

4.2.2 Control del sistema en posición

Usando el mismo bloque de control del apartado 4.1 con el nuevo modelo dinámico los resultados también son muy similares entre V-REP y Matlab, como puede observar en la Figura 22:

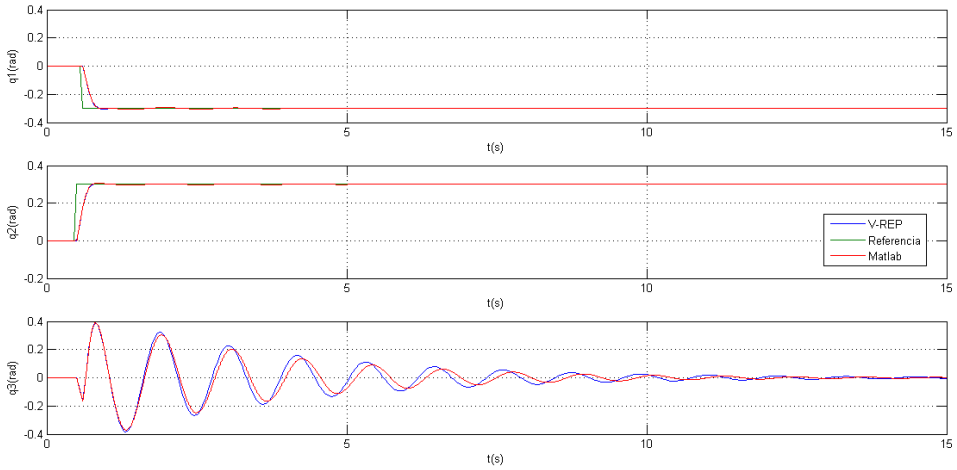

Figura 22: Control en posición del sistema 2dof con un muelle.

5 MODELADO Y CONTROL DEL BRAZO ALEXANDROS

En este capítulo se va a modelar y controlar el robot *Alexandros* [4], un brazo robótico flexible que cuenta con cuatro motores y cuatro muelles.

5.1 Análisis del sistema

En este apartado se va a llevar a cabo un análisis de un brazo robótico de cuatro grados de libertad flexibles (formados por un muelle y un motor) como el modelo de la Figura 23 donde se muestra a la izquierda la jerarquía y a la derecha el modelo 3D.

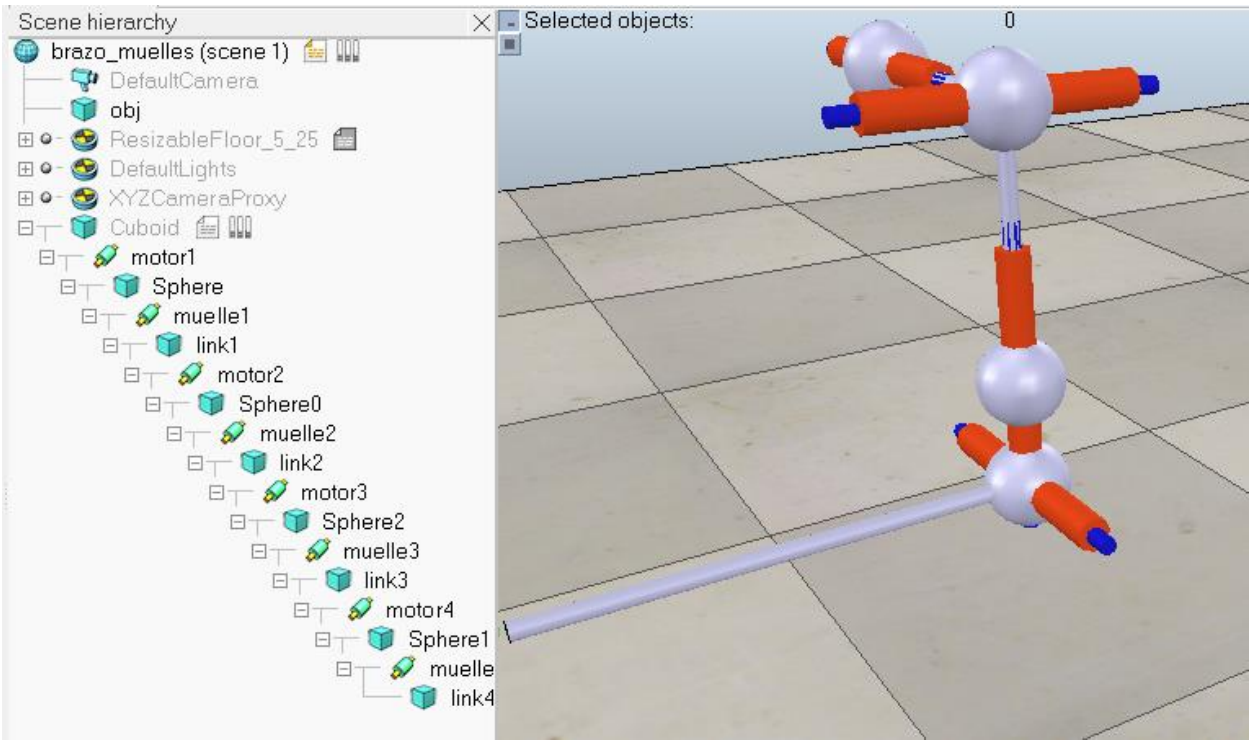

Figura 23: Modelo del brazo Alexandros.

Mirando la jerarquía de la Figura 23, los actuadores (motores) van a ser la variable de control. Están unidos mediante una esfera (sin masa) a un muelle que aporta flexibilidad y mediante un eslabón al siguiente motor. En la Figura 24 se muestran las propiedades dinámicas de los dos primeros eslabones y en la Figura 25 las propiedades referentes a los otros dos eslabones.

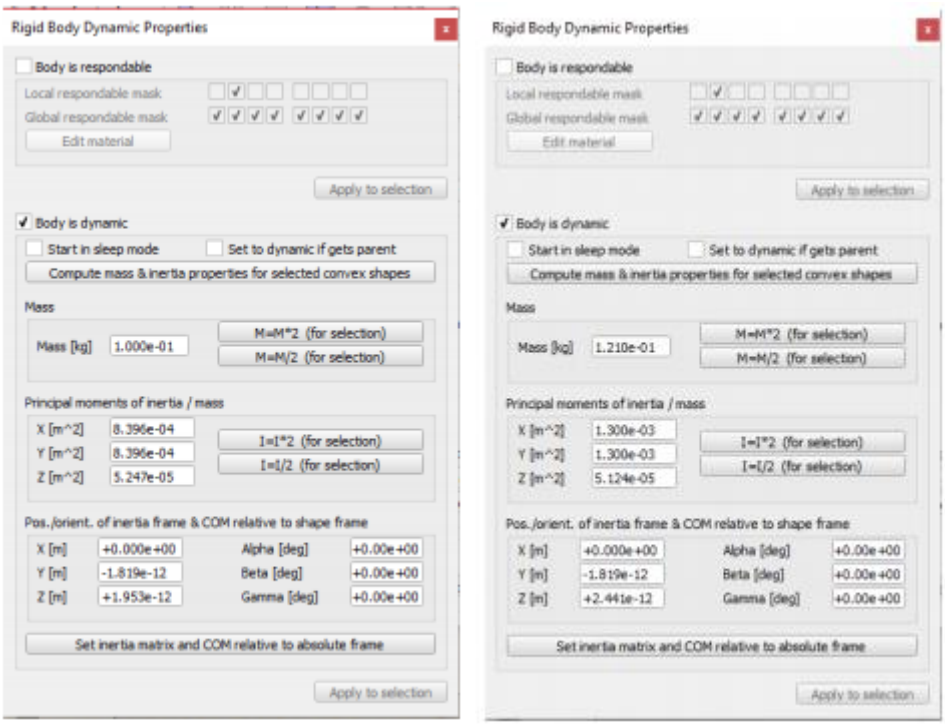

Figura 24: Propiedades dinámicas de los links 1 y 2.

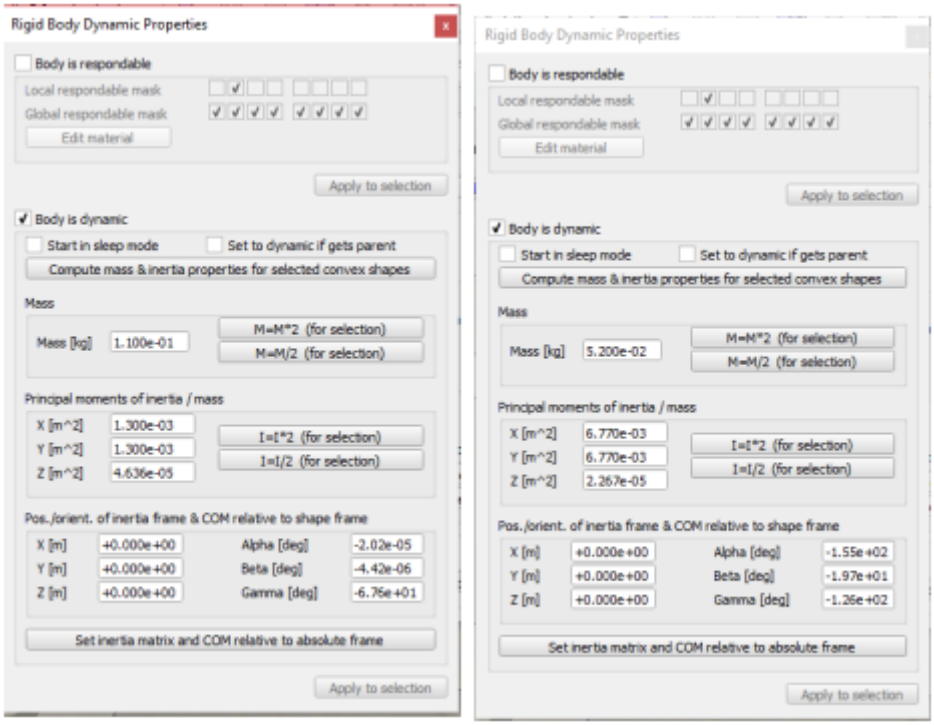

Figura 25: Propiedades dinámicas de los links 3 y 4.

5.2 Control de posición

En la Figura 26 se observa el esquema del control que se usar para el brazo. El bloque destacado de amarillo es el correspondiente a la sincronización con V-REP, entran las velocidades de los 4 motores y devuelve las posiciones angulares de los 4 motores y los cuatro muelles para realimentar al motro. Además también devuelva la posición del efector final para poder observar el error en posición. Los bloques Contact y ACCLIK se encargan de gestionar la acción de control del robot.

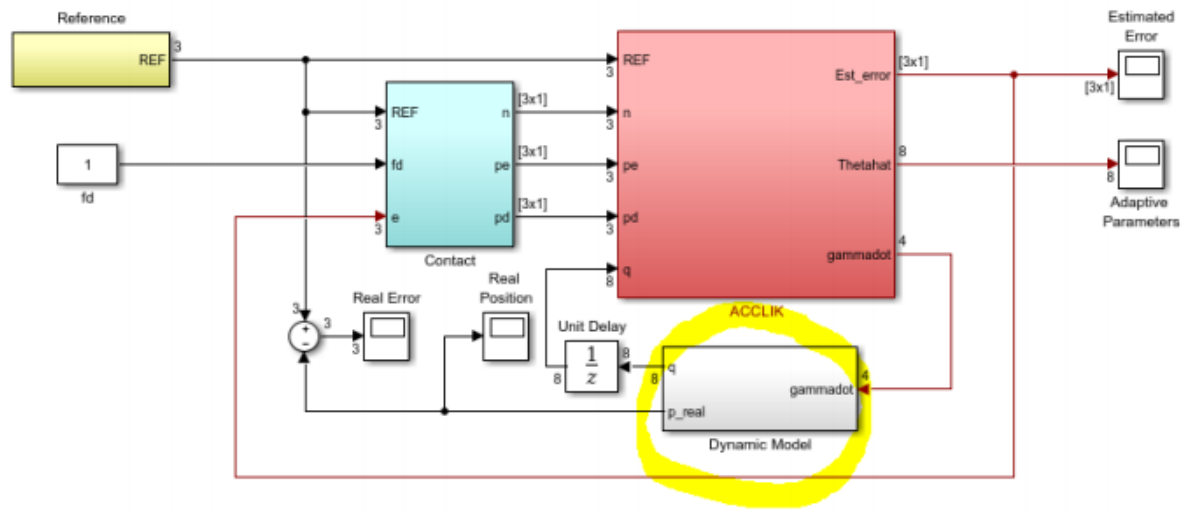

Figura 26: Diagrama de bloques de simulink

El siguiente script inicializa el sistema y lanza las simulaciones:

```
clear all;    clc
global modeR modeL L l_cg m g0 Gamma Gamma_p Kp K D Ds Ks tau_s Theta_0 ...
       ke q0 maxgdot freq_control Theta_h_max noise_factor contact_mode ...
       ARmin ARmax modeStairJOG
global clientID;     %VARIABLES GLOBALES PARA QUE LA FUNCIÓN step.m pueda
acceder
global j1;             %a ellas   y así no tenga que estar continu  amente
iniciando la comunicación
global j2;
global j3;
global j4;
global j5;
global j6;
global j7;
global j8;
global objeto;
%-------------------------------------------------------------------------%
%               RM PARAMETERS (definition of the manipulator)             %
%-------------------------------------------------------------------------%
modeR  = [  2       1       3       2      ];   % Rotation mode
modeL  = [  2       -3      -3      1      ];   % Link direction mode
L      = [  0.102   0.125   0.125   0.27   ];   % Link length
l_cg   = [  0.5     0.5     0.5     0.5    ];   % Link CG ratio
m      = [  0.101   0.121   0.110   0.052  ];   % Link mass
K      = diag([ 2.93 2.1 0.8 1.48 ]);            % RM stiffness matrix
ke     = 50;                                     % Contact entity stiffness
D      = diag([0.02 0.02 0.02 0.02]);            % RM damping matrix
tau_s  = [2 2 1 2];                              % Servo stall torque
maxgdot= 5;                                      % Maximum servo speed
Ds     = 1.5*diag(0.2*tau_s./maxgdot);           % Servo friction
q0     = 0*[-0.024 0 0 -0.047 0.024 0 0 0.047 ];  % Initial condition on q
```

```
g0     = 9.81;                                          % Gravity
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                  CONTROL GAINS, PARAMETERS AND MODES                    %
%-------------------------------------------------------------------------%
Kp           = 40*diag([0.75 0.8 0.9 0.55 ]);       % Proportional gains
Ks           = 20*0.016*diag([ 1 0.8 0.7 0.9 ]);  % Servo gains
Gamma        = 8e2*diag([10 10 0.1 1 1 1 0.1 1]); % Adaptive gains
Gamma_p      = diag([1 1 1 1 1 1 1 1]);           % Proj. hyperellipsoid
Theta_h_max  = 6;                                 % Proj. funct. maximum
Kaprox       = diag([   3   2   1   1.5  ]);      % First approach of K
Theta_0      = [zeros(4,1) ; diag(inv(Kaprox))];  % Initial adapt. cond.
freq_control = 50;                                % RM control frequency
noise_factor = 0;                                 % Noise sensor factor
contact_mode = 0;                                 % Contact enable/disable
modeStairJOG = 0;                                 % StairJOG enable/disable
ARmin        = 0.1;                               % Minimum I_JOG slope
ARmax        = 0.4;                               % Maximum I_JOG slope
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
disp('Program started');
Ts=0.02;
    vrep=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
    vrep.simxFinish(-1); % just in case, close all opened connections
    clientID=vrep.simxStart('127.0.0.1',19997,true,true,5000,5);

    if (clientID>-1)
        disp('Connected to remote API server');

        % enable the synchronous mode on the client:
        vrep.simxSynchronous(clientID,true);

        % start the simulation:
        vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait);
        %declarar los objetos (4 motores y 4 muelles)

[r1,j1]=(vrep.simxGetObjectHandle(clientID,'motor1#',vrep.simx_opmode_blocki
ng));

[r2,j2]=(vrep.simxGetObjectHandle(clientID,'motor2#',vrep.simx_opmode_blocki
ng));

[r3,j3]=(vrep.simxGetObjectHandle(clientID,'motor3#',vrep.simx_opmode_blocki
ng));

[r4,j4]=(vrep.simxGetObjectHandle(clientID,'motor4#',vrep.simx_opmode_blocki
ng));

[r5,j5]=(vrep.simxGetObjectHandle(clientID,'muelle1#',vrep.simx_opmode_block
ing));

[r6,j6]=(vrep.simxGetObjectHandle(clientID,'muelle2#',vrep.simx_opmode_block
ing));

[r7,j7]=(vrep.simxGetObjectHandle(clientID,'muelle3#',vrep.simx_opmode_block
ing));

[r8,j8]=(vrep.simxGetObjectHandle(clientID,'muelle4#',vrep.simx_opmode_block
ing));
```

```
[r9,objeto]=(vrep.simxGetObjectHandle(clientID,'obj#',vrep.simx_opmode_block
ing));
       %ejecuta la simulación en simulink
        sim('Alexandros_Simulator_VREP');

        % stop the simulation:
        vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait);

        % Now close the connection to V-REP:
        vrep.simxFinish(clientID);
    else
        disp('Failed connecting to remote API server');
    end
    vrep.delete(); % call the destructor!
```

El script que se comunica con V-REP, formando el bloque de "Dynamic model":

```
function [ out ] = step( in )
            vel1=in(1); %Variables de entrada (velocidad de cada motor)
            vel2=in(2);
            vel3=in(3);
            vel4=in(4);
global clientID;    %variables globales que le llegan desde SINCRO.m
global j1;
global j2;
global j3;
global j4;
global j5;
global j6;
global j7;
global j8;

            vrep=remApi('remoteApi');
           %Fijar las velocidades de los motores (acción de control)
            vrep.simxSetJointTargetVelocity(clientID,       j1,       vel1,
vrep.simx_opmode_oneshot_wait);
            vrep.simxSetJointTargetVelocity(clientID,       j2,       vel2,
vrep.simx_opmode_oneshot_wait);
            vrep.simxSetJointTargetVelocity(clientID,       j3,       vel3,
vrep.simx_opmode_oneshot_wait);
            vrep.simxSetJointTargetVelocity(clientID,       j4,       vel4,
vrep.simx_opmode_oneshot_wait);

            %Lee la posición angular de los motores
            [s1,        pos1]=vrep.simxGetJointPosition(clientID,       j1,
vrep.simx_opmode_streaming);
            [s2,        pos2]=vrep.simxGetJointPosition(clientID,       j2,
vrep.simx_opmode_streaming);
            [s1,        pos3]=vrep.simxGetJointPosition(clientID,       j3,
vrep.simx_opmode_streaming);
            [s2,        pos4]=vrep.simxGetJointPosition(clientID,       j4,
vrep.simx_opmode_streaming);
            %Lee la posición angular de los muelles
            [s1,       posm1]=vrep.simxGetJointPosition(clientID,       j5,
vrep.simx_opmode_streaming);
            [s2,       posm2]=vrep.simxGetJointPosition(clientID,       j6,
vrep.simx_opmode_streaming);
            [s1,       posm3]=vrep.simxGetJointPosition(clientID,       j7,
vrep.simx_opmode_streaming);
```

```
            [s2,       posm4]=vrep.simxGetJointPosition(clientID,       j8,
vrep.simx_opmode_streaming);


                vrep.simxSynchronousTrigger(clientID);


out=[double(pos1),double(pos2),double(pos3),double(pos4),double(posm1),doubl
e(posm2),double(posm3),double(posm4)];

end
```

5.3 Control de fuerza

Para llevar a cabo un control de la fuerza aplicada, se ha diseñado un sistema como el de la Figura 27 en el que se han puesto dos paredes con un muelle, como el analizado en el capítulo 3, que las une. El brazo empujará la pared de la derecha y al estar conectadas por un muelle se ejercerá una fuerza. Para controlar esta fuerza ejercida (en régimen permanente), se va a suponer que son conocidas tanto la constante K del muelle como la posición exacta de la pared en reposo. Usando la ley de Hooke:$Fuerza = K * desplazamiento$, se puede calcular que desplazamiento provoca la fuerza que estamos buscando y aplicarlo a la posición de reposo de la pared. En este caso la constante K tiene un valor de 10 N/m.

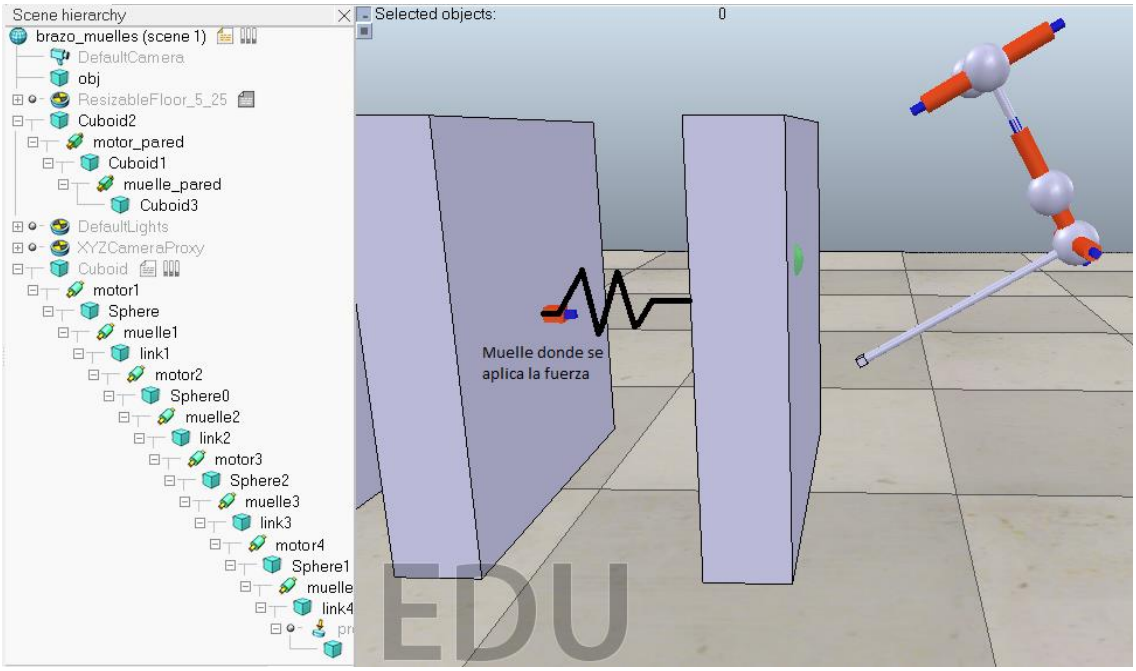

Figura 27: Modelo completo del control de fuerza.

El problema que plantea el control en fuerza que se plantea es que es un control en bucle abierto, si no se tiene un modelo exacto el control va a fallar. Una posible forma de solucionar este problema sería realimentando en fuerza y con algún tipo de control PID, pero queda fuera del alcance de este proyecto.

5.3.1 Control de fuerza sin errores de modelado

En este primer experimento se va a considerar que el modelo es perfecto y no hay ningún tipo de error. En la Figura 28 se muestra la respuesta del sistema a una referencia de 0.15N y en la Figura 29 se lleva a cabo el mismo experimento pero con una referencia de 0.25N. En ambos casos hay una oscilación inevitable debida a la concatenación de elementos dinámcos (muelles del brazo y la pared) pero siempre queda en torno al punto deseado.

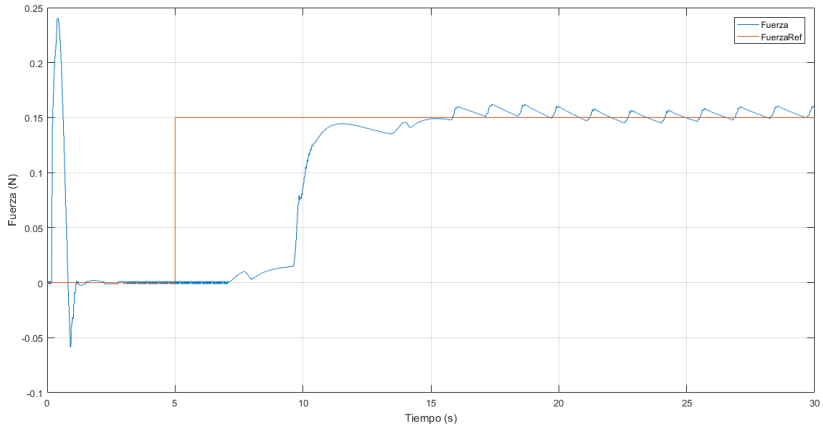

Figura 28: Control de fuerza con referencia de 0.15N

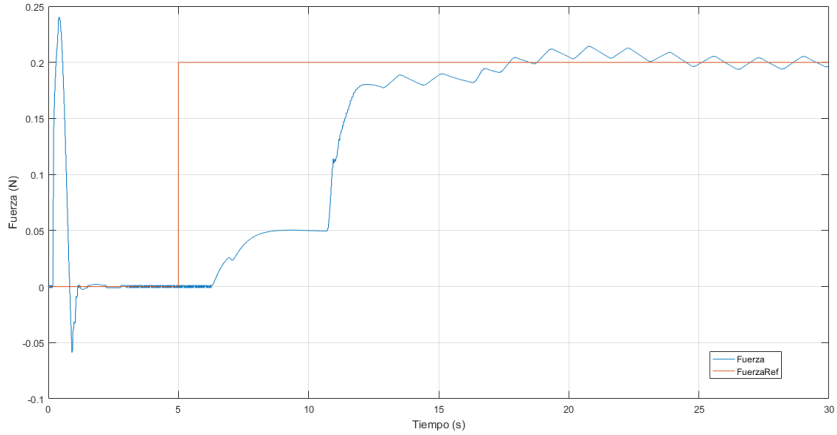

Figura 29: Control de fuerza con referencia de 0.15N

5.3.2 Control de fuerza con errores de modelado en el muelle

En la Figura 30 se aumenta la constante del muelle que se le proporciona el control, siendo el muelle real menos rígido de lo esperado y como es de esperar el brazo no alcanza la referencia de fuerza con el desplazamiento calculado internamente. En la Figura 31 ocurre el caso contrario, el muelle es más rígido de lo que se prevee y por tanto ejerce una fuerza mayor.

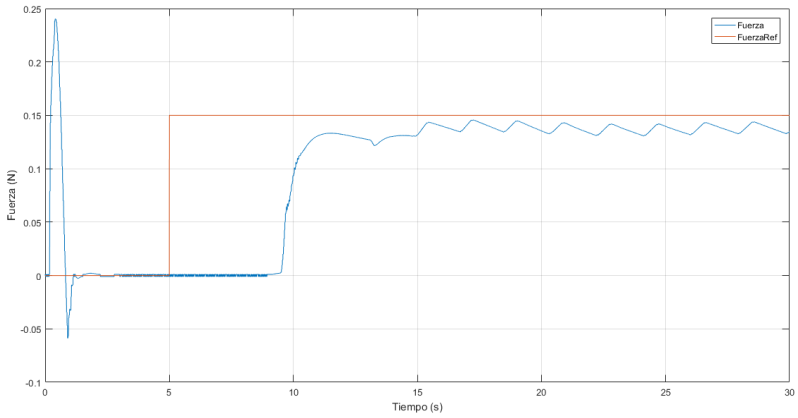

Figura 30: Control de fuerza con la constante k del muelle errónea (12)

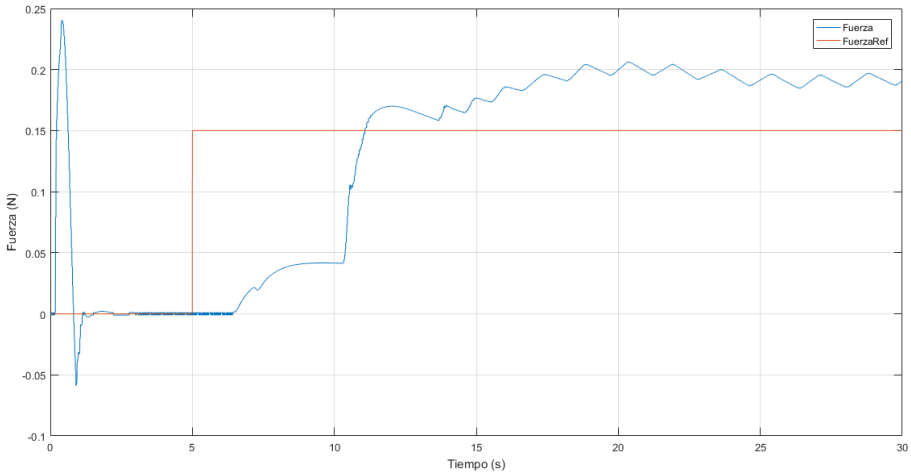

Figura 31: Control de fuerza con la constante k del muelle errónea (8)

5.3.3 Control de fuerza con errores de modelado en la posición inicial

En este último experimento no se posiciona el brazo en la posición de reposo en la que debería estar. En la Figura 32 se muestra como se ejerce más fuerza de la esperada debido a que empieza en una posición más cercana a la pared, y con un mismo desplazamiento que en el caso sin errores de modelado comprime más el muelle.

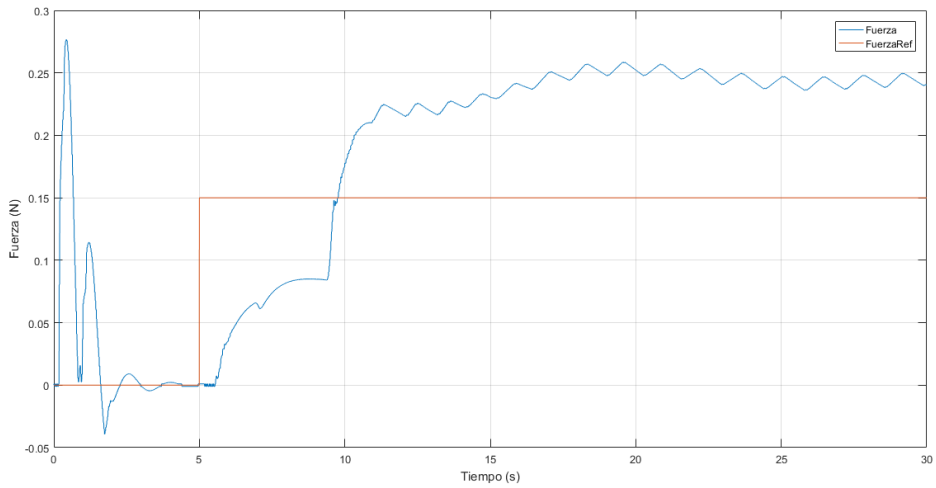

Figura 32: Control de fuerza con posición de reposo errónea

6 MODELADO Y CONTROL DE UN UAV

En este capítulo se va a analizar en primer lugar las hélices propias de V-REP. Al ver el resultado de los experimentos, se procederá a crear una hélice en LUA. Después se modelará un UAV con estas hélices y por último se le implementará un control Backstepping.

6.1 Análisis de las hélices para ser usadas en el UAV

En primer lugar, antes de analizar el UAV completo se van a estudiar las hélices con las que construir el UAV con el objetivo de entender bien su funcionamiento individualmente.

6.1.1 Elementro Propeller en V-REP

Por defecto V-REP incorporta un elemento hélice para las simulaciones. Para comprender que hace se va a analizar su comportamiento y ver si es viable para un drone o si por el contrario merece más la pena crear uno desde cero en Lua dentro de V-REP. El elemento tiene la apariencia de la Figura 33 y tiene un script en Lua para que ejerza un empuje y cree unas partículas meramente visuales.

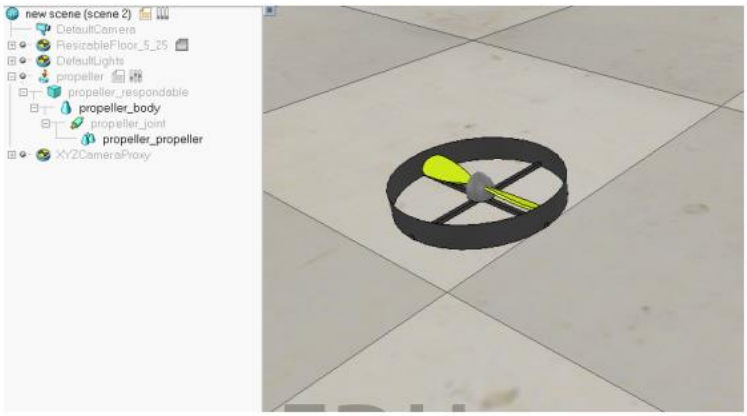

Figura 33: Hélice por defecto en V-REP

El funcionamiento de estas hélices se basa en algunos parámetros físicos con algunos valores que no quedan claro, solo hay un parámetro que tiene sentido modificar denominado *particles velocity*. Modificando este parámetro como se observa en la Figura 34 se puede calcular el empuje realizado a partir de la aceleración (tomando 0 como caída libre). Se ve una tendencia claramente lineal, por tanto podemos deducir que no es un modelo fiable para una simulación compleja.

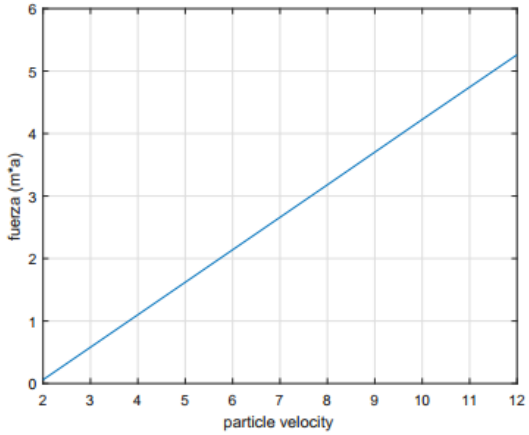

Figura 34: Relación fuerza frente a particle velocity

6.1.2 Hélice creada basada en un modelo aerodinámico

Viendo que el modelo que viene por defecto en V-REP no es muy fiel a una hélice real se ha creado un modelo en V-REP. Tiene la apariencia de la Figura 35 y está formado por un motor, un cilindro de base y otro que hace de pala meramente visual. Ambos cilindros tienen masas e inercias despreciables y la fuerza y el momento creados al girar la pala se crean mediante un script que se encuentra en el cilindro base.

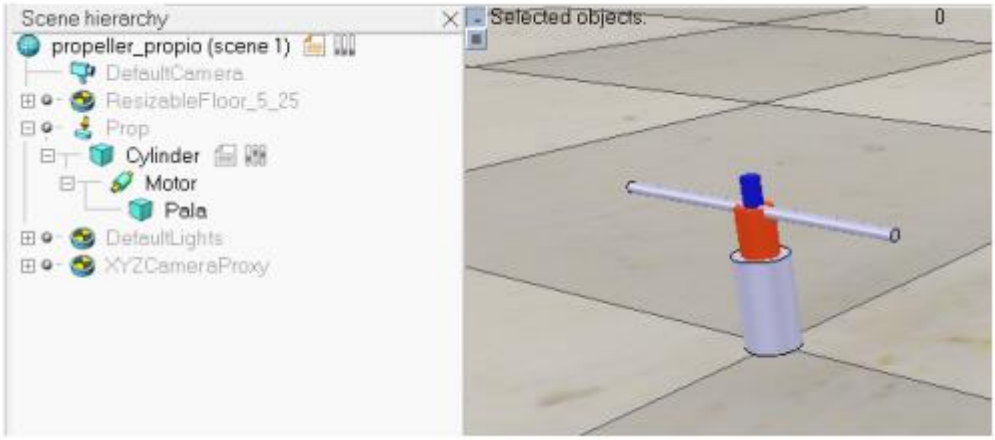

Figura 35: Hélice creada desde cero en V-REP

Tras consultar sobre modelos propulsivos se decide implementar el siguiente:

```
--Units used in this script correspond to the IS criteria.

if (sim_call_type==sim_childscriptcall_initialization) then
propellerRespondable=simGetObjectHandle('Cylinder')
propellerJoint=simGetObjectHandle('Motor')
body=simGetObjectAssociatedWithScript(sim_handle_self)
ts=simGetSimulationTimeStep()
pos_I0=simGetObjectPosition(propellerRespondable,-1)
-- Rotor parameters ----------------------------------
Radius=simGetScriptSimulationParameter(sim_handle_self,'Radius')
AirDensity=simGetScriptSimulationParameter(sim_handle_self,'AirDensity')
Chord=simGetScriptSimulationParameter(sim_handle_self,'Chord')
Blades=simGetScriptSimulationParameter(sim_handle_self,'Blades')
theta_0=simGetScriptSimulationParameter(sim_handle_self,'theta_0')
Cl_alpha=simGetScriptSimulationParameter(sim_handle_self,'Cl_alpha')
Cd0=simGetScriptSimulationParameter(sim_handle_self,'Cd0')
k_Cd=simGetScriptSimulationParameter(sim_handle_self,'k_Cd')
-- Rotor auxiliar parameters--------------------------
Area=math.pi*Radius^2 --(m^2)
sigma=Blades*Chord/(math.pi*Radius) --solidity of the rotor (dimensionless)
```

```
end



if (sim_call_type==sim_childscriptcall_actuation) then



-- Rotor variables--------------------------------
Omega = simGetJointTargetVelocity(propellerJoint)-- Rotor Angular Velocity (rad/s)

v_I= simGetVelocity (body)-- UAV speed (m/s)/ angular rates (rad/s) (inertial frame)
v_Ix=(v_I[1]) --velocity (x) stimate (m/s)
v_Iy=(v_I[2])
v_Iz=(v_I[3])
Rmatrix=simGetObjectMatrix(propellerRespondable,-1)-- Transformation matrix (body to inertial)
Rmatrix_x=Rmatrix[9]
Rmatrix_y=Rmatrix[10]
Rmatrix_z=Rmatrix[11]
--Aerodinamic variables--------------------------
v_Bz=v_Ix*Rmatrix_x+v_Iy*Rmatrix_y+v_Iz*Rmatrix_z-- Vertical component of the speed in the body frame
Vaero=math.sqrt(v_Ix^2+v_Iy^2+v_Iz^2)--Aerodinamic velocity
alpha_r=math.asin(v_Bz/(Vaero+1e-6))--Rotor angle of attack

--Propulsive Variables----------------------------
Vaero_vtip=Vaero/(Omega*Radius)-- Aerodinamic velocity to rotor tip velocity ratio
mu=Vaero_vtip*math.cos(alpha_r)-- Dimensionless forward speed
nu=Vaero_vtip*math.sin(alpha_r)-- Dimensionless axial speed
k1=sigma*Cl_alpha/8 -- auxiliar parameter
k2=k1*theta_0*(2/3+mu^2)-- auxiliar parameter

--The following steps correspond to the solution of the quartic that results from using a BEMT
--model. Thus, to understand their meaning it is recommended to study the generic solution of
--a quartic.

--Resolution Parameters Step I-------------------
A1=-nu
A2=Vaero_vtip^2-k1^2
A3=nu*mu^2+k1*k2
A4=nu^2*mu^2-k2^2
```

```
--Resolution Parameters Step II------------------
b1=0.5*(A1^2+A2-1);
b2=A3-b1;
b3=0.5*(A2*A4-A1^2*A4-A3^2);

--Resolution Parameters Step III-----------------
c1=b1^2-3*b2
c2=2*b1^3-9*b1*b2+27*b3
c3=(0.5*(c2+math.sqrt(c2^2-4*c1^3)))^(1/3)

--Resolution Parameters Step IV------------------
d1=-(b1+c3+c1/c3)/3
d2=math.sqrt(d1^2-A4)
d3=(d1-A3)/d2

--Solution of the quartic
auxsol=A1*d3-2*d2
if auxsol<0 then sign_auxsol=-1
else sign_auxsol=1 end
x=0.5*(-A1-d3*sign_auxsol+math.sqrt(A1^2+d3^2-4*d1+2*math.abs(auxsol)))

--Thrust results--
CT=2*(k2-k1*x) --Thrust coefficient (dimensionless)
T=AirDensity*Area*(Omega*Radius)^2*CT-- Rotor thrust (N)
CQ=sigma*(Cd0+k_Cd*theta_0^2)*(1+mu^2)/8+2*k1*(theta_0/3-x/2)+0.25*sigma*k_Cd*x*(x-
4*theta_0/3) --Torque coefficient (dimensionless)
T_B= CQ*AirDensity*Area*(Omega*Radius)^2*Radius; -- Rotor torque (Nm)

force_B={0,0,T} --thrust in body frame
torque_B={0,0,T_B}--torque in body frame
Rmatrix[4]=0
Rmatrix[8]=0
Rmatrix[12]=0
force_I=simMultiplyVector(Rmatrix,force_B)--thrust in inertial frame
torque_I=simMultiplyVector(Rmatrix,torque_B)--torque in inertial frame
simAddForceAndTorque(propellerRespondable,force_I,torque_I)
simSetFloatSignal("T",v_Bz/(Vaero+1e-6))
```

```
end


if (sim_call_type==sim_childscriptcall_sensing) then



end


if (sim_call_type==sim_childscriptcall_cleanup) then



end
```

Este modelo aerodinámico es bastante más complejo que el anterior, sin embargo no es válido en casos de caídas bruscas, de todas maneras no se busca que el UAV sufra una caída brusca. En la Tabla 1 se muestran los parámetros que usa el script. Para crear un script en V-REP: Seleccionar el elemento al que se se le asociará el script → Add →Associated child script →Non threaded.

Parámetro	Valor	Unidades
Radius	0.2032	m
AirDensity	1.225	kg/m^3
Chord	0.025	m
Blades	4	number
Cl_alpha	5.7	dimensionless
theta_0	0.1412	rad

Tabla 1: Parámetros del script de Lua para la hélice.

6.2 Análisis del cuadrótor usando las hélices creadas en V-REP

Con cuatro hélices como las mencionadas en el apartado anterior y algunos elementos pasivos se ha modelado el cuadrótor de la Figura 36. Cabe destacar que tiene un pequeño cubo gris, que no aporta nada dinámicamente pero está alineado con los ejes de coordenas y evita usar un cambio de ejes en el controlador. También se le ha añadido desde Matlab una pequeña dinámica a las hélices que no se contempla en el script anterior (tiempo de subida 0.3 segundos) para un comportamiento más realista.

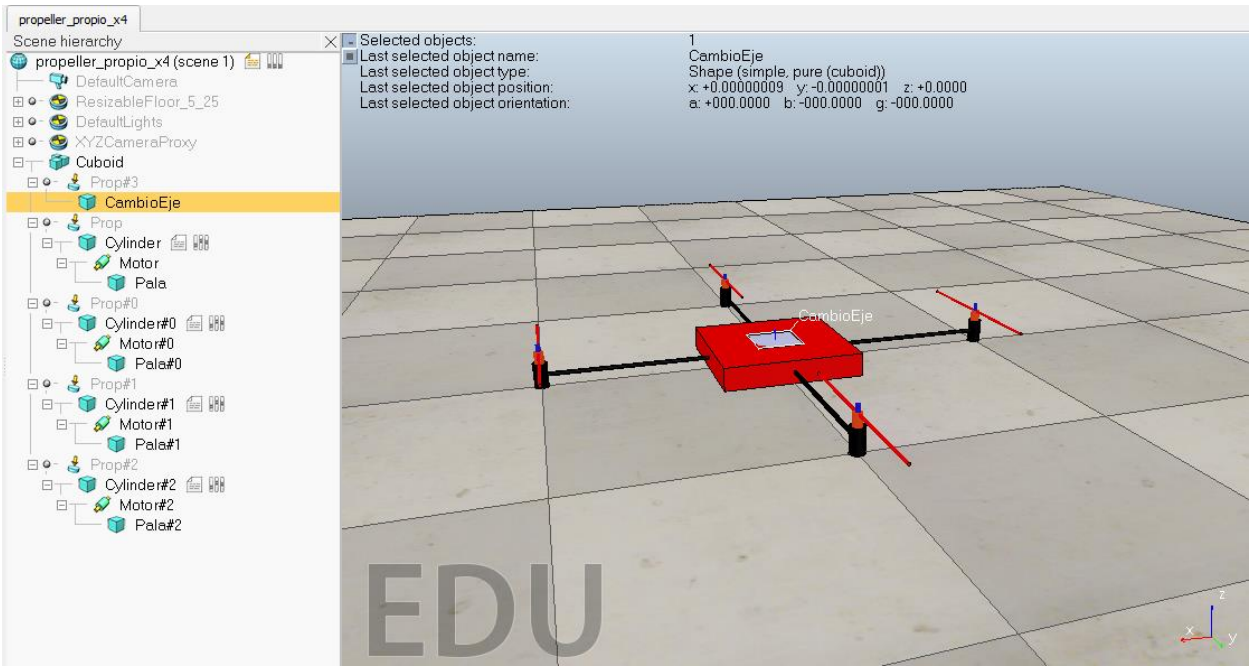

Figura 36: Jerarquía y modelo del UAV

Para comprobar que el modelo de V-REP es correcto y no tiene errores (sobretodo los scripts en Lua), se van a establecer las velocidades de cada rótor a 750 rad/s y dejar que el sistema evolucione en bucle abierto sin ningún tipo de control. En la Figura 37 se comparan tres sistemas: un modelo de Matlab, un modelo de V-REP sin la dinámica en las hélice mencionada anteriormente y otro de V-REP sin dináminca en las hélices. Se observa como el modelo de Matlab y el de V-REP sin dinámica coinciden con un error mínimo y el modelo de V-REP con dinámica tiene un retraso aproximadamente del tiempo de subida que se estábleció.

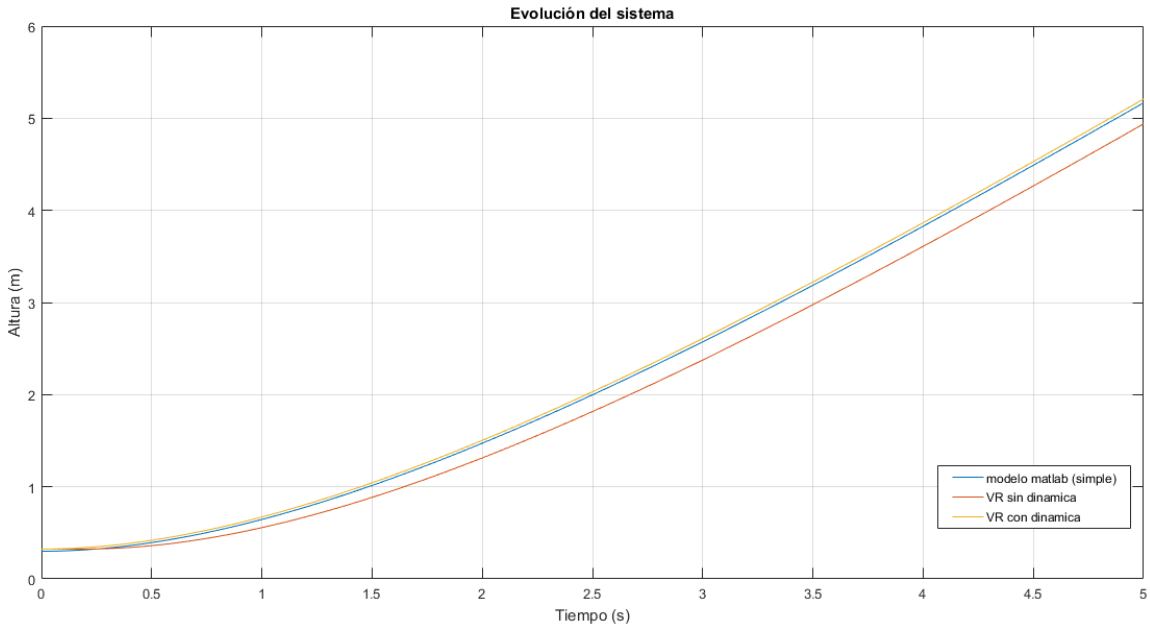

Figura 37: Comparativa de los sistemas en bucle abierto.

Para unir las hélices al cuerpo del cuadrotor se usan sensores de fuerza a los cuales no se les da uso como tal. Para insertarlos: Add→Force Sensor. En la Figura 38 se muestran las propiedades dinámicas (masa e inercias) del UAV.

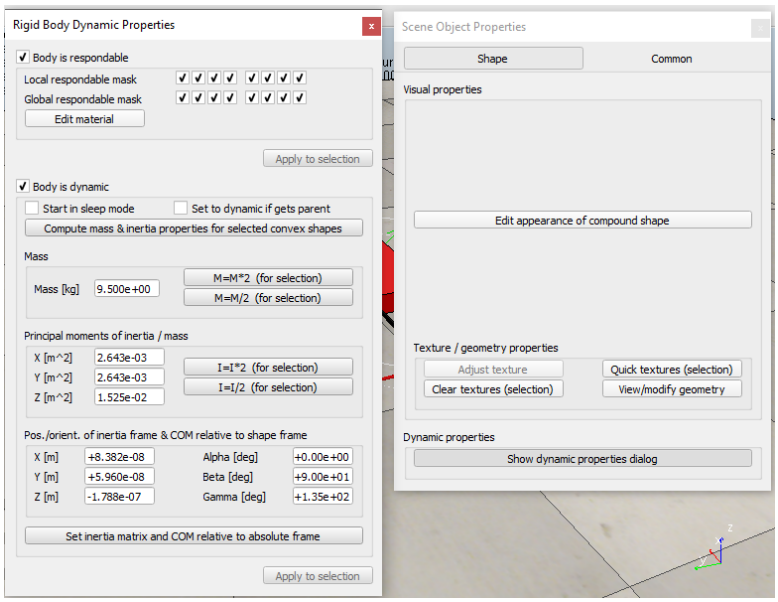

Figura 38: Propiedades dinámicas del UAV.

El siguiente script permite inicializar la sincronización y ejecutar Simulink y V-REP:

```
clear all;
%close all;
global clientID;%VARIABLES GLOBALES PARA QUE LA FUNCIÓN step.m pueda acceder
global cuerpo1;
global motor1;
global motor2;
global motor3;
global motor4;

disp('Program started');
Ts=0.05;
vrep=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
vrep.simxFinish(-1); % just in case, close all opened connections
clientID=vrep.simxStart('127.0.0.1',19997,true,true,5000,5);

if (clientID>-1)
    disp('Connected to remote API server');

    % enable the synchronous mode on the client:
    vrep.simxSynchronous(clientID,true);

    % start the simulation:
    vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait);
    %declarar los objetos

[~,cuerpo1]=(vrep.simxGetObjectHandle(clientID,'Cuboid#',vrep.simx_opmode_bl
ocking));

[~,motor1]=(vrep.simxGetObjectHandle(clientID,'Motor#',vrep.simx_opmode_bloc
king));

[~,motor2]=(vrep.simxGetObjectHandle(clientID,'Motor#2#',vrep.simx_opmode_bl
ocking));

[~,motor3]=(vrep.simxGetObjectHandle(clientID,'Motor#0#',vrep.simx_opmode_bl
ocking));

[~,motor4]=(vrep.simxGetObjectHandle(clientID,'Motor#1#',vrep.simx_opmode_bl
ocking));
```

```
   %ejecuta la simulación en simulink
    sim('propulsion');

    % stop the simulation:
    vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait);

    % Now close the connection to V-REP:
    vrep.simxFinish(clientID);
else
    disp('Failed connecting to remote API server');
end
vrep.delete(); % call the destructor!
disp('Program ended');
```

Función que implementa la comunicación:

```
function [ out ] = step( in )
          omega1=in(1); %velocidad de los ejes 1 y 2 que llegan desde simulink
          omega2=in(2);
          omega3=in(3);
          omega4=in(4);
global clientID;    %variables globales que le llegan desde SINCRO.m
global cuerpo1;
global motor1;
global motor2;
global motor3;
global motor4;

vrep=remApi('remoteApi');
%Fijar la velocidad de cada hélice
vrep.simxSetJointTargetVelocity(clientID,        motor1,        omega1+1e-5,
vrep.simx_opmode_oneshot_wait);
vrep.simxSetJointTargetVelocity(clientID,        motor2,        omega2+1e-5,
vrep.simx_opmode_oneshot_wait);
vrep.simxSetJointTargetVelocity(clientID,        motor3,        omega3+1e-5,
vrep.simx_opmode_oneshot_wait);
vrep.simxSetJointTargetVelocity(clientID,        motor4,        omega4+1e-5,
vrep.simx_opmode_oneshot_wait);
%%%%% HAY QUE SUMAR 1e-5 PARA EVITAR ERRORES NÚMERICOS%%%%%%%%

%Leer posición y orientación
[~,posicion]=vrep.simxGetObjectPosition(clientID,                cuerpo1,-1,
vrep.simx_opmode_streaming);
[~,orientacion]=vrep.simxGetObjectOrientation(clientID,          cuerpo1,-1,
vrep.simx_opmode_streaming);

posx=posicion(1);
posy=posicion(2);
posz=posicion(3);
alpha=orientacion(1);
beta=orientacion(2);
gamma=orientacion(3);



    vrep.simxSynchronousTrigger(clientID);
```

```
out=[double(posx),double(posy),double(posz),double(alpha),double(beta),doubl
e(gamma)];

end
```

Por último en la Figura 39 se muestra el diagrama de bloques de simulink para llevar a cabo este experimento, en el bloque "V-REP" se llama a la función descrita anteriormente.

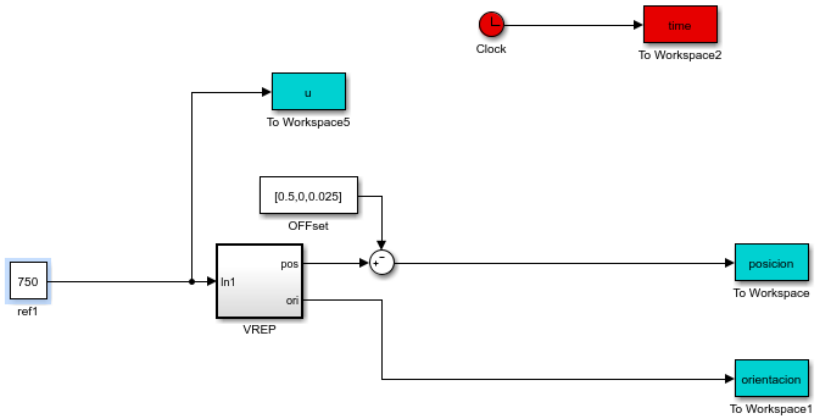

Figura 39: Diagrama de bloques de simulink para el UAV en bucle abierto

6.3 Implementación de un control Backstepping para el UAV

Para poder controlar el UAV se ha implementado un control Backstepping [5] cuya explicación queda fuera del alcance del proyecto. En la Figura 40 se observan los dos primeros bloques del control. En el primero se obtiene la referencia (posición comandada) y se calculan también la velocidad y aceleración comandadas. En el segundo, a partir de la posición, velocidad, las referencias y la integral del error calcula el empuje necesario para subir y la posición de los ángulos roll y pitch del UAV para que llegue a la posición deseada. En la Figura 41 se muestra el control de actitud, que a partir de los ángulos roll y pitch comandados por el control de posición y los ángulos y velocidades actuales del UAV calcula el empuje necesario para que los ángulos lleguen a la referencia. Por último en la Figura 42 primero se observa un bloque que a partir del empuje (T) y el par (Tau) que necesita el UAV calcula las velocidades angulares de las hélices (U). Estas velocidades angulares serán la acción de control que se enviará a V-REP, para ello se usa el bloque de la derecha, que es la sincronización con V-REP. Además aquí se incluye un pequeño filtro para simular la dinámica de las hélices.

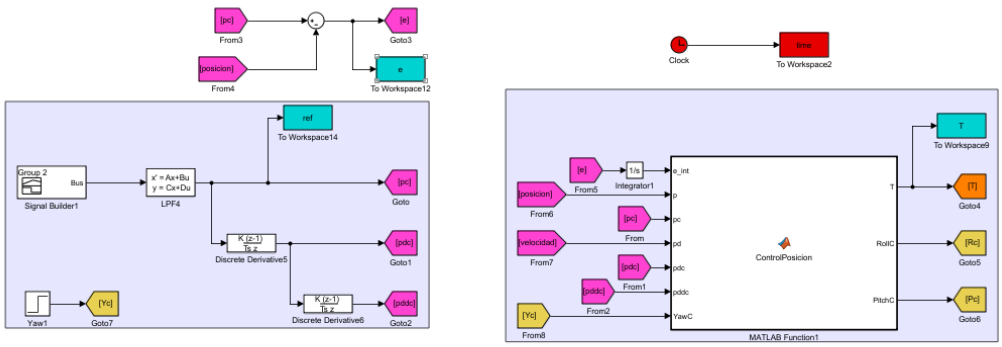

Figura 40: Control Backstepping (1/3)

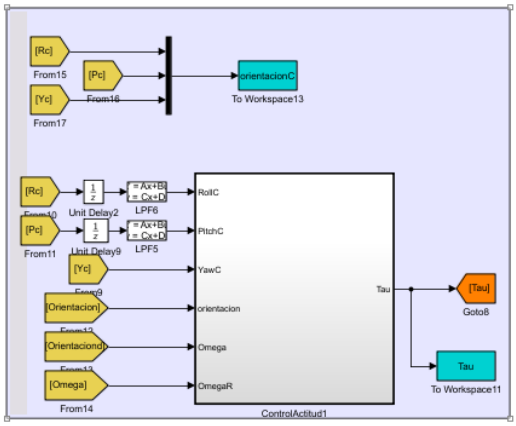

Figura 41: Control Backstepping (2/3)

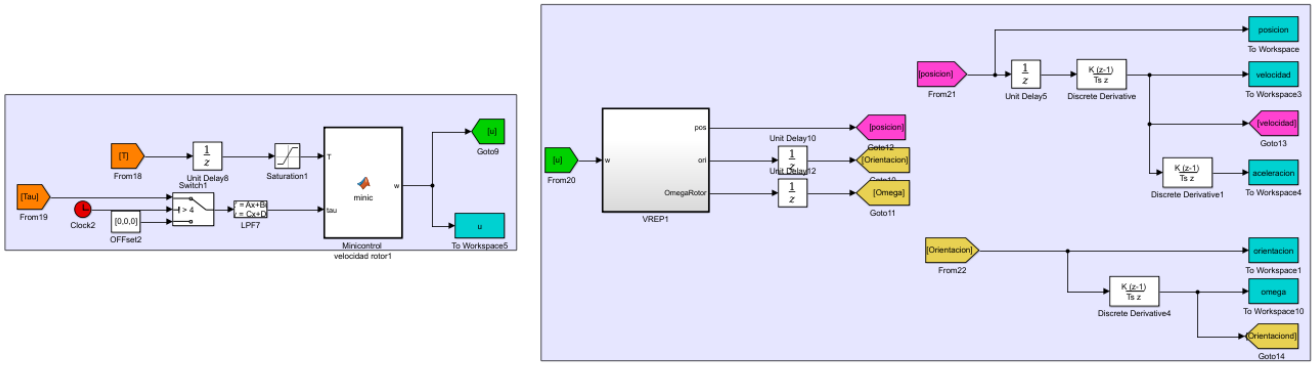

Figura 42: Control Backstepping (3/3)

El código para la comunicación entre Simulink y V-REP es el mismo del apartado anterior y el código para sintonizar los parámetros del control e inicializar ambas simulaciones es el siguiente:

```
clear all;
close all;
global clientID;     %VARIABLES GLOBALES PARA QUE LA FUNCIÓN step.m pueda
acceder
global cuerpo1;
global motor1;
global motor2;
global motor3;
global motor4;

%variables del control
global Kd Kp Ki g m Gamma Gamma_2_P l k b TOmega Gamma_1_P IUAV Irotor
Gamma2i;
disp('Program started');
Ts=0.05;
vrep=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
vrep.simxFinish(-1); % just in case, close all opened connections
clientID=vrep.simxStart('127.0.0.1',19997,true,true,5000,5);

if (clientID>-1)
    disp('Connected to remote API server');

    % enable the synchronous mode on the client:
```

```
    vrep.simxSynchronous(clientID,true);

    % start the simulation:
    vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait);
    %declarar los objetos

[~,cuerpo1]=(vrep.simxGetObjectHandle(clientID,'CambioEje#',vrep.simx_opmode
_blocking));

[~,motor1]=(vrep.simxGetObjectHandle(clientID,'Motor#',vrep.simx_opmode_bloc
king));

[~,motor2]=(vrep.simxGetObjectHandle(clientID,'Motor#2#',vrep.simx_opmode_bl
ocking));

[~,motor3]=(vrep.simxGetObjectHandle(clientID,'Motor#0#',vrep.simx_opmode_bl
ocking));

[~,motor4]=(vrep.simxGetObjectHandle(clientID,'Motor#1#',vrep.simx_opmode_bl
ocking));
%variables globales para el control:

Kd=diag([0.4 0.4 5])*2/3; %posicion
Kp=diag([0.3 0.3 4])*2.3;
Ki=diag([1.2 1.2 0.05])*0.03;
g=9.81;
m=10.305;
b=(1.9468e-04)/4;
l=0.5;
Rrotor=0.2032;
rho=1.225;
A=pi*Rrotor^2;
k=0.015*rho*A*Rrotor^2;

IUAV= 9.500e+00*diag([2.643e-03 2.643e-03 1.525e-02]);
Irotor= 4.750e-04*1.131e-03;


Gamma=-0*diag([2 2 0.9]);
Gamma_2_P=-9*1*diag([-4 4 2]); % ganancia proporcional angulos
Gamma2i=Gamma_2_P*0.1; %unkwown
TOmega=5*diag([15 15 15]);
Gamma_1_P=8*diag([2 2 0.9]); %ganancia "integral" angulos

   %ejecuta la simulación en simulink
    sim('propulsion_clara');

    % stop the simulation:
    vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait);

    % Now close the connection to V-REP:
    vrep.simxFinish(clientID);
else
    disp('Failed connecting to remote API server');
end
vrep.delete(); % call the destructor!
disp('Program ended');
```

6.4 Pruebas de vuelo

En la Figura 43 se muestra un primer experimento en el cual se eleva un metro hacia arriba el UAV, después se desplaza10 cm positivos en el eje Y y 20cm negativos en el mismo eje. En la Figura 44 asciende hasta la misma altura que el experimento anterior y realiza dos desplazamientos positivos en el eje X de 10cm. Tiene un comportamiento bastante aceptable aunque podría refinarse aún más el control.

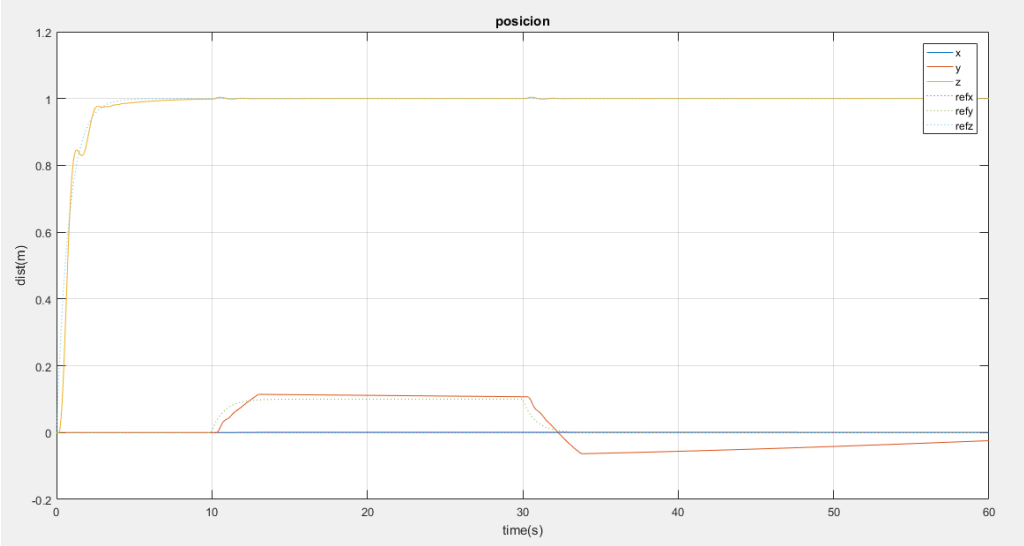

Figura 43: Experimento uno del control del UAV.

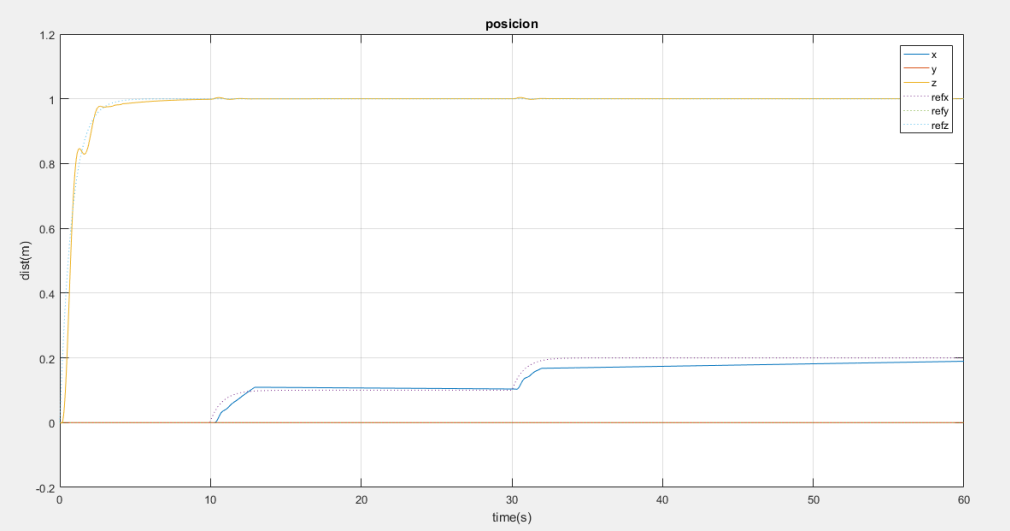

Figura 44: Experimento dos del control del UAV.

REFERENCIAS

[1] https://es.mathworks.com/products/matlab.html

[2] http://www.coppeliarobotics.com/

[3] http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm

[4] A. Suarez, A. E. Jimenez-Cano, V. M. Vega, G. Heredia, A. Rodriguez-Castan˜o, and A. Ollero, "Design of a lightweight dual arm system for aerial manipulation," Mechatronics, vol. 50, pp. 30 – 44, 2018. [Online]. Available: http://www.sciencedirect.com/science/

[5] R. de Cos, Carlos, Acosta, José Ángel, Ollero Baturone, Anibal: Command-Filtered Backstepping Redesign for Aerial Manipulators Under Aerodynamic and Operational Disturbances. Pag. 817-828. En: Advances in Intelligent Systems and Computing. Springer, Cham. 2018. ISBN 978-3-319-70833-1